Forecasting Algorithms for Intelligent Resource Scaling: An Experimental Analysis

Yanlei Diao*
Amazon Web Services
yanleid@amazon.fr

Dominik Horn*
Amazon Web Services
domhorn@amazon.de

Andreas Kipf
UTN
andreas.kipf@utn.de

Oleksandr Shchur
Amazon Web Services
shchuro@amazon.de

Ines Benito
Amazon Web Services
benitoin@amazon.fr

Wenjian Dong
Amazon Web Services
wjdong@amazon.fr

Davide Pagano
Amazon Web Services
dpagano@amazon.com

Pascal Pfeil
Amazon Web Services
pfeip@amazon.de

Vikram Nathan
Amazon Web Services
vrnathan@amazon.com

Balakrishnan Narayanaswamy
Amazon Web Services
muralibn@amazon.com

Tim Kraska
Amazon Web Services
timkrask@amazon.com

ABSTRACT

There has been a growing demand for making modern cloud-based data analytics systems cost-effective and easy to use. AI-powered intelligent resource scaling is one such effort, aiming at automating scaling decisions for serverless offerings like Amazon Redshift Serverless. The foundation of intelligent resource scaling lies in the ability to forecast query workloads and their resource consumption accurately. Although the forecasting problem has been extensively studied across various domains, there is a lack of thorough analysis of existing forecasting algorithms for large-scale, real-world cloud query workloads. This paper fills this gap by providing an in-depth analysis of forecasting algorithms for real-world cloud workloads, covering the fundamental data characteristics that distinguish query workload forecasting from prior problems and evaluating the strengths and limitations of existing algorithms in this new domain. We anticipate that our findings will provide valuable insights in informing the design of an efficient and effective solution for production use, as well as in steering the forecasting community toward more effective algorithms of high real-world impact.

*Lead authors.

SoCC '24, November 20–22, 2024, Redmond, WA, USA

ACM ISBN 979-8-4007-1286-9/24/11.
https://doi.org/10.1145/3698038.3698564

CCS CONCEPTS

• **Information systems → Database design and models**; • **Computing methodologies → Ensemble methods**.

ACM Reference Format:
Yanlei Diao, Dominik Horn, Andreas Kipf, Oleksandr Shchur, Ines Benito, Wenjian Dong, Davide Pagano, Pascal Pfeil, Vikram Nathan, Balakrishnan Narayanaswamy, and Tim Kraska. 2024. Forecasting Algorithms for Intelligent Resource Scaling: An Experimental Analysis. In *ACM Symposium on Cloud Computing (SoCC '24), November 20–22, 2024, Redmond, WA, USA.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3698038.3698564

1 INTRODUCTION

There has been a growing demand for making modern cloud-based data analytics systems cost-effective and easy to use. Intelligent scaling in Amazon Redshift is one such effort: While Redshift's *provisioned clusters* allow the user to specify the cluster size and the underlying EC2 instance type to provision, *serverless clusters* automatically provision and scale the data warehouse capacity to respond to changing customer workloads. In particular, the new intelligent scaling feature boosts the existing serverless offering by intelligently allocating compute resources, based on AI technology, in the face of a variety of workload variability [26].

Table 1: Categorizing different practical use cases of workload and resource forecasts

Use Case	Forecast Target	Dimensionality	Long/Short-term Forecast	Scaling Operation
Idle Windows	Idle period	Univariate	Long/short-term	Release unused resources ; schedule maintenance or disruptive operations
Dynamic Compute	High load (spikes)	Univariate / Multivariate	Short-term	Allocate additional compute resources
Reserved Compute	Entire workload	Multivariate (w. distributions)	Long-term	Optimize reserved capacity

The foundation of such intelligent resource scaling lies in the ability to accurately forecast query workloads and their resource consumption. Although the forecasting problem has been extensively studied across domains such as finance, weather, and sales, with techniques ranging from statistical methods [4, 6, 16] to recent deep learning models [5, 9, 18, 27, 29, 38] to the latest foundation models [3, 10, 39], there is a lack of thorough analysis of existing forecasting algorithms for large-scale, real-world cloud query workloads.

For query workload and resource forecasting, Table 1 summarizes three practical, important use cases. In particular, the first use case refers to the forecast of idle windows, the second refers to the forecast of high load windows, and the third refers to the long-term forecast of resource needs. All these use cases may occur within a compute cluster for intelligent resource scaling. Details are as follows.

Use Case 1: *Idle Windows.* Serverless offerings are usually billed on a per-use basis, i.e., customers only pay when they actively use their endpoint. A typical online optimization is to release unused resources during extended idle periods. Forecasting such time windows of low to no activity is therefore crucial for an economic implementation. Another important capability unlocked by forecasting idle windows is the ability to schedule maintenance or other disruptive operations more optimally.

Use Case 2: *Dynamic Compute.* Acquiring new compute resources usually takes a certain amount of time, i.e., there is a delay between the request and fulfillment. Re-actively requesting resources might not be sufficient in the face of short but pronounced spikes in the workload, e.g., the case that a workload burst is already mostly over by the time the additional compute becomes available to use. For such cases, it is desirable to have a *predictive* approach that foresees workload bursts and proactively allocates enough additional compute capacity to adequately deal with them once they do arrive. In cloud warehouses, a workload burst can for example be characterized as a period in the future with a large number of arriving queries (e.g. many short dashboard queries), large concurrent memory needs (e.g. a few memory-intensive ETL queries), a total compute usage exceeding the available resources, or a combination of these factors. Contrary to the previous use case that targets idle periods, the forecast here focuses on workload peaks, specifially their magnitude.

Use Case 3: *Reserved Compute.* As mentioned above, acquiring compute dynamically comes at the cost of additional overhead. It is therefore desirable to have some reserved amount of compute resources always ready to go. Determining the correct capacity to reserve requires forecasting the workload on a longer horizon (e.g., next couple of days). For systems such as Amazon Redshift, one option here is to perform multivariate forecast on the number of queries, total resource needs (e.g., sum of required CPU time), and concurrent memory needs. Further having distributions on these forecast signals will help enable scaling decisions with high confidence.

Contributions. This paper provides an in-depth analysis of forecasting algorithms for predicting real-world cloud workloads. In particular, in the context of Redshift Serverless or more specifically its next-generation AI scaling [26]. Our study is motivated by the observation that, while forecasting has been well studied in other domains, query workload forecasting has been under-addressed in the literature, and the challenges faced by the practitioners are not well known to the research community. To this end, our analysis characterizes the fundamental data properties that distinguish the forecasting problem in this new domain from other well-studied forecasting problems, and highlights the strengths and limitations of existing forecasting algorithms for this unique problem. The insights gained in the above analysis inspire us to design our own forecasting algorithms to better suit the query workload forecasting problem. More specifically, our paper makes the following contributions:

1) *Data Analysis.* We begin by analyzing the fundamental data characteristics of cloud workload metadata history, such as spikiness and multiple seasonal periods. These patterns are prevalent in production clusters, yet the exact mix of spikiness and seasonality patterns in each cluster varies significantly. In addition, we observe that serverless workloads exhibit higher degrees of spikiness and volatility (irregularity) than provisioned workloads, making them more challenging to forecast.

2) *Baseline Model Analysis.* We then analyze the "predictability" of production workloads using a baseline algorithm (seasonal naive weekly) that forecasts the future workload based on the observations in the past week, which has been shown to be a suitable baseline given the data characteristics. Our analysis shows that *this baseline can forecast*

light query loads within a factor of 2 for about 60%-80% of the clusters (depending on the time series of choice), but is subject to severe performance degradation in forecasting high-load windows (which is required in use cases such as dynamic compute).

3) *Advanced and Custom Models.* We further investigate a range of machine learning-based forecasting algorithms, from statistical to deep learning-based to foundation models, for forecasting high load windows. Our findings reveal that no individual machine learning model consistently outperforms the baseline, and their forecasting accuracy falls short of satisfactory levels. Therefore, we develop customized ensemble models, both local and global, that combine multiple promising candidate models using automatically learned weights within an AutoML framework (specifically, AutoGluon [32]). Notably, even the predetermined model ensembles in AutoGluon prove inadequate for our datasets. To address this, we introduce, to the best of our knowledge, the first custom ensemble models for query workload forecasting based on a systematic approach. These models are specifically tailored to overcome the challenges presented by our unique forecasting problem. Across the fleet, our custom ensemble models have been shown to provide remarkable improvements over the baseline: the ensemble models *reduce 38-77% errors on average for the serverless workloads and 12-56% errors on average for the provisioned workloads.*

4) *Future Research Directions.* The results of our analysis also enable us to identify several promising directions for future research on forecasting. We anticipate that our findings will be valuable in steering the forecasting community towards developing more effective algorithms of high real-world impact, especially for query workload forecasting and intelligent resource scaling.

To facilitate the research community in conducting similar analyses, we have released in the public domain a dataset of both provisioned and serverless workloads (https://github.com/amazon-science/redset/). This dataset contains all the necessary features and offers valuable properties such as diverse workload sampling, enabling similar studies to ours.

2 RELATED WORK

We survey a number of areas broadly related to our work.

Forecasting Methods. Time series forecasting has been intensively studied in statistics and machine learning literature. We survey methods in a few categories below.

Statistical models. Traditional statistical models such as ETS [16], ARIMA [6] and Theta [4] are designed to capture common time series patterns like trend or seasonality. These models often make unrealistic assumptions such as Gaussian distribution of errors or stationarity, which can lead to inaccurate forecasts when these assumptions are violated [8]. Still, statistical models often provide competitive results and remain widely used by practitioners [15, 21].

Deep models. Recent years have seen a surge of deep learning models for forecasting [5, 9]. Popular examples of this class of models include DeepAR [29], MQCNN [38], PatchTST [27] and Temporal Fusion Transformer [18]. The models themselves are based on deep learning architectures for sequential data such as recurrent neural networks, 1D convolutional neural networks, or transformers. The above deep learning models still need to be trained from scratch for each specific task, which makes them not suitable for cases when limited historic data is available.

Foundation models. Pre-trained time series models like TimesFM [10], Moirai [39] and Chronos [3] represent the most recent advances in the field of time series forecasting. These models are pre-trained on large collections of time series data and can generate forecasts for new, unseen time series in a zero-shot manner.

Robust, Fast Online Time Series Analysis. Motivated by cloud use cases, a recent line of work has focused on extending classical time series forecasting methods such as STL (Seasonal-Trend decomposition using LOESS) with improved functionality, robustness, and efficiency on data streams. RobustPeriod [36] finds multiple periods and variable lengths of the periods via wavelet transform. However, it finds peaks in the time domain and takes the mean value as the period length, which may not suit our diverse, spiky workloads. FastRobustSTL [37] is an improved version of RobustSTL [35] for efficient computation. OnlineSTL [24] supports stream processing for STL decomposition. OneShotSTL [13] is an online algorithm for STL decomposition with lowest complexity. As we will show in our study, none of the individual forecasting models work well for complex product workloads. For this reason, prior work that focuses only on the STL family of algorithms is unlikely to be effective for fleet-wide workloads, which differ widely in data characteristics and the patterns to detect.

DBMS Workload Forecasting has gained significant interest in the database community recently. Peloton [28] is an envisioned self-driving DBMS that operates on forecast query workloads. Its workload forecasting solution, QueryBot 5000 [19], predicts the expected arrival rate of queries in the future. To this end, it extracts the templates from query strings, clusters the query templates based on their arrival rate temporal patterns, and then predicts the future arrival rate pattern for each query cluster. Similarly, Tiresias [1] predicts data access characteristics such as latency and volume of accessed data based on query arrival patterns. Meduri et al. proposed to predict the next query based on the current query and forecasts the literals in the query as a bin of possible values using RNNs and Q-learning techniques [23]. Sibyl [14] further extends to forecasting future query traces with precise query statements for time-evolving workloads, with the underlying assumptions that real workloads are highly recurrent and highly predicable.

The above solutions do not suit our intelligent resource forecasting problem for a number of reasons: First, prior work such as Sibyl [14] focuses on specialized workloads, such as point lookup queries in a decision support system and analytical queries in an internal revenue reporting system. While recurrence and predictability may be stated for these workloads, in this study we re-examine such statements over fleet-wide production clusters. In particular, production workloads are very diverse and often a mix of recurring and ad-hoc workloads. Intelligent resource scaling requires accurate forecasts of the entire mix, not a subset of the workloads. Further, even recurring queries may appear at different times, with a significant time shift across days or weeks, which makes it difficult to have fine-grained resource forecasting (e.g., for hourly or 10-minute windows). Second, by design our forecaster does not require access to query strings. This is because extracting query templates, clustering these templates, and building a forecasting model for each query cluster incurs substantial overhead on production clusters. We prefer a more efficient time series forecasting solution to minimize model training overhead on production clusters.

Cloud Workload Analysis. Shahrad et al. [31] analyzed the workload of Function as a Service (FaaS), in particular, a 2-week dataset of invocations of Azure Functions including their trigger types, invocation frequencies and patterns, and resource needs. This dataset includes limited recurring functions: 95% of applications have at most 10 functions and only 0.04% of the applications have more than 100 functions. In comparison, our analysis used 3-month datasets that include much larger query workloads and exhibit a wide variety of spikiness and seasonality patterns. Furthermore, the method for modeling inter-invocation times in that study is not feasible in our case given our large number of queries and the fact that our forecasts need to accommodate both recurring and ad-hoc queries. WiseFuse [20] further analyzed a two-week trace of Azure Durable Functions and reported that the vast majority of serverless workflows are recurring: the top 5% most frequent workflows constitute 94.6% of workflow invocations. In contrast, query workloads analyzed in our study are less regular due to the presence of ad-hoc queries and the varying arrival times of recurring queries. Recent work [33] analyzed the Amazon Redshift fleet as a whole, showing that write-heavy data pipelines are prominent and workloads vary over time in both load and type. However, that study did not articulate or quantify the challenges in workload forecasting, which is the focus of our paper.

3 FLEET DATA ANALYSIS

To support all of the use cases mentioned in earlier sections, the intelligence of forecasting lies in a deep dive into the metadata history of past queries, specifically their runtimes and resource consumption metrics. When a significant fraction of the workload recurs, the forecaster can leverage the recurring patterns to predict the future workload in a given horizon and quantify the confidence in such predictions.

In this section, we provide an in-depth analysis of production cluster workloads, including their intrinsic patterns and impact on effective workload forecasting.

Repetition Rate. For workload analysis, a relevant notion is repetition rate, that is, how many queries repeat in each client cluster. Recent work has shown that many production workloads exhibit recurring patterns. For example, 50% SQL queries repeat in about 50% of Amazon Redshift clusters [33].

Despite the above results, we posit in this work that "repeatability" is not a sufficient measure for precise resource forecasting. The reasons are two-fold: (1) Even if a fraction of queries repeat in a client cluster, their arrival times may vary. For example, a recurring ETL job may occur with half an hour difference across days, which makes it hard to achieve precise resource forecasting for small time windows, e.g., 10 minute horizons. (2) The entire workload on a cluster is a mix of recurring queries and ad-hoc queries. Resource forecasting needs to capture the query runtime, memory consumption, etc. for the entire workload, not just the recurring queries.

Predictability. For the above reasons, we introduce a new notion, "predictability", to evaluate real-world production workloads. Predictability is defined based on *the accuracy of forecasting a resource metric (e.g., the number of queries, total resource consumption, or the concurrent memory needed) into a future horizon.* When the accuracy of a forecasting algorithm is sufficiently high (e.g., within a factor of 2 of the ground truth value), we claim that this cluster's workload is predictable.

In this study, we conduct an analysis of the fleet of Redshift clusters on "predictability". The results of this study will provide valuable information to answer two main questions: (1) What data characteristics of query workloads make the resource forecasting problem unique and perhaps more challenging compared to well-studied domains like weather and sales forecasting? (2) Given such data characteristics, how many production workloads can be considered "predictable" given available forecasting methods?

3.1 Characteristics of Real-world Workloads

Our analysis uses two datasets comprised of internal telemetry data, especially query metadata such as arrival time, execution time, maximum memory needed, the number of tables

Table 2: Data characteristics of hourly time series in provisioned (A) workloads.

	Provisioned Clusters														
	Query Count (QC)					Runtime Total (RT)					Bytes Scanned (BS)				
	Spkness	$C(24)$	$C(168)$	$C(336)$	$C(672)$	Spkness	$C(24)$	$C(168)$	$C(336)$	$C(672)$	Spkness	$C(24)$	$C(168)$	$C(336)$	$C(672)$
p50	1.00	0.55	0.52	0.45	0.36	1.05	0.44	0.42	0.33	0.26	1.08	0.52	0.46	0.36	0.30
p90	3.78	0.89	0.87	0.83	0.79	5.38	0.86	0.83	0.74	0.64	3.69	0.90	0.86	0.81	0.73
p100	8.28	0.99	1.0	1.0	1.0	29.7	0.95	0.99	0.98	0.97	12.5	0.98	0.99	1.0	1.0
Mean	**1.70**	0.52	0.51	0.46	0.38	**2.52**	0.43	0.43	0.37	0.30	**1.88**	0.51	0.46	0.41	0.33

Table 3: Data characteristics of hourly time series in serverless (B) workloads.

	Serverless Clusters														
	Query Count (QC)					Runtime Total (RT)					Max Memory (MM)				
	Spkness	$C(24)$	$C(168)$	$C(336)$	$C(672)$	Spkness	$C(24)$	$C(168)$	$C(336)$	$C(672)$	Spkness	$C(24)$	$C(168)$	$C(336)$	$C(672)$
p50	2.76	0.48	0.42	0.28	0.19	2.94	0.37	0.28	0.22	0.13	2.79	0.37	0.27	0.20	0.13
p90	7.20	0.87	0.78	0.73	0.64	6.68	0.83	0.76	0.67	0.56	5.76	0.84	0.78	0.74	0.64
p100	30.6	0.98	0.97	0.97	0.98	20.0	0.98	0.96	0.98	0.96	44.5	0.96	0.97	0.97	0.97
Mean	**3.96**	0.47	0.40	0.40	0.25	**3.76**	0.42	0.36	0.30	0.22	**3.60**	0.41	0.36	0.30	0.23

scanned, the number of bytes scanned, etc. Both datasets were extracted from workload histories of production clusters, with up to 3 months of query metadata per workload.

Datasets. The two selected datasets have different yet representative characteristics for resource forecasting.

Workload A: Provisioned Clusters. Although intelligent resource scaling is an Amazon Redshift Serverless feature, we include Redshift's provisioned workload in our study to understand whether there is currently a difference in workload between customers selecting Redshift Serverless over provisioned clusters. This dataset includes workload metrics of 200 Redshift provisioned clusters for 3 months in 2024. These clusters were sampled from the Redshift fleet using customized stratified sampling, with more weights given to busier clusters in terms of the number of queries and accumulated query runtime.

Workload B: Serverless Clusters. Our second dataset contains 3 month of workload traces of 200 clusters from Redshift Serverless at the end of 2023. Again, they were sampled from the fleet using our customized sampling method.

Signals for Resource Forecasting. As informed by our use cases, we computed a variety of signals useful for resource scaling decisions, each of which is a time series for forecasting. These signals include: (*i*) *Query Count* (QC) is the number of queries in each window. (*ii*) *Runtime Total* (RT) is the sum of the query runtime of all the queries in the window, which is a rough approximation of the CPU resources needed by these queries. As we don't have access to the CPU cycles consumed by queries in our datasets, we use the sum of query runtime here as an approximation. (*iii*) *Max Memory* (MM) is the maximum concurrent memory needed by the queries in a window. (*iv*) *Bytes Scanned* (BS) is the total number of bytes scanned by the queries in a window.

As a default setting, we bucket metrics into fixed hourly windows for time series generation. Particular use cases may use smaller window sizes to generate higher frequency time series, which we will present in a later study. Some of the signals are available in one dataset but not the other. We generated as many signals as each dataset allows. In particular, for the provisioned dataset, we generated QC, RT, and BS time series, while for the serverless dataset, we generated QC, RT, and MM time series.

Quantitative Measures. In this study, we analyze both datasets using the following measures:

Spikiness. The most salient property of our datasets is spikiness: the value of a time series (e.g., the number of queries) can change from a low value in a time unit (e.g., an hourly query count equals 0) to a very high value in the next unit (e.g., caused by the arrival of a large batch of queries in the next hour). To capture this trend, we adopt the spikiness measure from a recent paper [33] but modify the normalization term to better suit our study.

Formally, let $X = \{X_1, X_2, \ldots, X_t, \ldots, X_n\}$ be a univariate time series of size n. The spikiness is defined to be the root mean squared difference between two consecutive points, further normalized by the mean μ of the time series.

$$Spk(X) = \frac{1}{\mu}\sqrt{\frac{1}{n-1}\sum_{t=1}^{n-1}(X_{t+1} - X_t)^2} \tag{1}$$

Seasonalities. In the time series literature, seasonality is defined using autocorrelation with different lags [15]. More specifically, seasonality of period length l is defined using autocorrelation with lag l, with μ and σ^2 referring to the mean and variance of the time series, respectively.

$$C(X, l) = \frac{1}{(n-1)\sigma^2}\sum_{t=1}^{n-l}(X_t - \mu)(X_{t+l} - \mu) \tag{2}$$

To capture seasonality at different granularities, we compute the autocorrelation with lag $l = 24$ (a seasonal period of 24 hours, i.e., a daily pattern), $l = 168$ (a seasonal period of 168 hours, i.e., a weekly pattern), $l = 336$ (a bi-weekly pattern), and $l = 672$ (a 4-week pattern, as an approximation of the monthly pattern).

EXPT3_1: Spikiness Analysis. Tables 2 and 3 summarize the spikiness and autocorrelation scores of the provisioned and serverless datasets, respectively. Regarding spikiness, our main observations are as follows:

Most notably, serverless clusters have higher spikiness scores for all of its three time series than those of the provisioned clusters. For the serverless dataset, the mean spikiness across all client clusters is in the range of 3.60 - 3.96 for the three time series, and the 90th percentile (p90 value) of the spikiness is in the range of 5.76 to 7.20. In comparison, for the provisioned dataset, the mean spikiness ranges from 1.70 - 2.55 and the p90 value ranges from 3.78 to 5.38. Figure 1 shows example time series, plotted at hourly and daily frequencies, for the provisioned dataset (Figure 1(a)-1(b)) and the serverless dataset (Figure 1(c)-1(d))[1]. These time series were chosen from the top-10 clusters in each respective dataset in terms of spikiness in the test data (where the last two weeks are reserved as the test data, as marked by the red dashed line in the plots). However, individual provisioned clusters can also exhibit spiky patterns, as illustrated in Figure 1(e)-1(f) — this is why even for provisioned clusters, Redshift offers a feature called concurrency scaling that aims to absorb these spiky bursts of queries.

Furthermore, we observe that certain time series exhibit extremely spiky patterns in about 10% of clusters, as evidenced by the significant increase in the spikiness score from the 90th to the 100th percentile: these time series include provisioned RT series, serverless QC series, serverless RT series, and serverless MM series.

These characteristics present a significant challenge for forecasting algorithms in capturing spikes. The problem is further compounded by the fact that some spikes are recurring but with time shifts (as shown by the peaks during the second and third weeks from the end of the timeline in Figure 1(d)), while others are ad-hoc, representing unexpected changes (as depicted by a standalone peak in the fifth week in Figure 1(d)).

EXPT3_2: Seasonality Analysis. We further observe diverse seasonalities in our datasets. As Tables 2 and 3 show, for our hourly time series, autocorrelation of lag 24 and that of lag 168 tend to have significant values from p90 to p100, indicating the strong presence of daily and weekly patterns in about 10% of clusters.

[1]All the time series plots in this paper have the y-axis normalized by the maximum value of a specific cluster. The x-axis contains the time delta from the start of the test set (in days).

However, each individual cluster can have very different autocorrelation scores for different lags. Figures 1(a)-1(f) show examples of different seasonalities. For provisioned cluster P1, lag 24 has the highest score, indicating a daily pattern. In contrast, serverless cluster S2 has the highest correlation score for the weekly pattern, although the patterns in the last three weeks do not agree on the count of the spikes or the exact timing of their occurrences within the week. Furthermore, provisioned cluster P3, as shown in Figure 1(e)-1(f), has the highest correlation scores for the bi-weekly pattern $C(336)$ and the 4-week pattern $C(672)$. As we can see from the daily time series plot in Figure 1(f), there is a high peak every four weeks and a medium-sized peak halfway between two high peaks, while for most weeks, the load is higher on the first few days of a week and lower on the weekend, but with exceptions.

Finally, the autocorrelation scores (p50, p60, etc.) in the serverless clusters tend to be lower than those in the provisioned clusters. This indicates less regular workloads in serverless clusters, which is expected as clients with less regular workloads tend to opt for serverless offerings as such workloads benefit from the pay-per-use billing model.

Summary. We now summarize our observations as follows:

(1) *Spikiness* is the prominent feature of query workload datasets, especially pronounced in the serverless dataset.
(2) *Diverse seasonalities* are prevalent in our datasets while the exact set of relevant patterns (daily, weekly, bi-weekly, or monthly) vary significantly from cluster to cluster.
(3) *Serverless workloads* exhibit more spiky patterns yet are less regular than provisioned workloads, hence expected to be harder to predict.

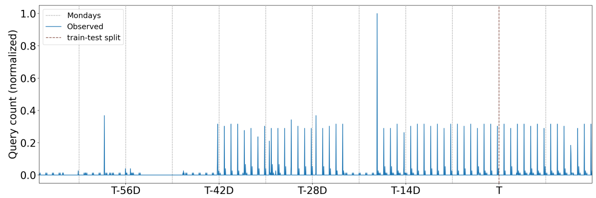
All of these observations call for effective forecasting algorithms to capture the diverse, complex patterns in cloud query workloads.

3.2 Predictability of Production Workloads

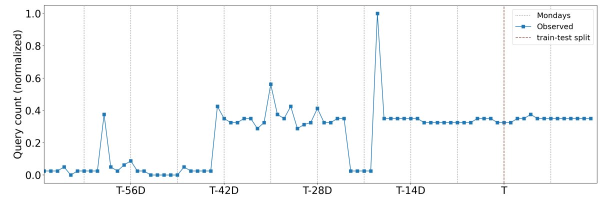
In this study, we analyze the above two datasets for the "predictability" of their query workloads. Recall that predictability in our study is defined based on the accuracy of forecasting a resource metric time series: roughly speaking, if the accuracy is within a factor of 2 of the ground truth, we claim that the resource metric of this cluster is predictable. Below, we will introduce our experimental setup and the specific error metric in use.

Experimental Setup. We assess the predictability of query workloads in the setting of 7-day forecast, e.g., to forecast the hourly time series for the upcoming 7 days.

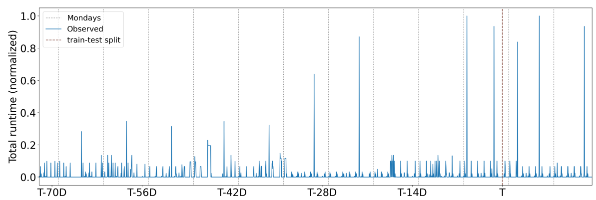
Baseline Algorithm. Our evaluation in this section uses baseline algorithms as informed by the data analysis above.



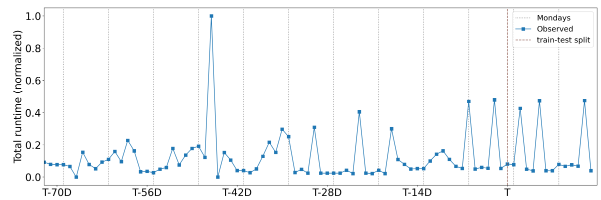
(a) Provisioned cluster P1: hourly query count (QC) series



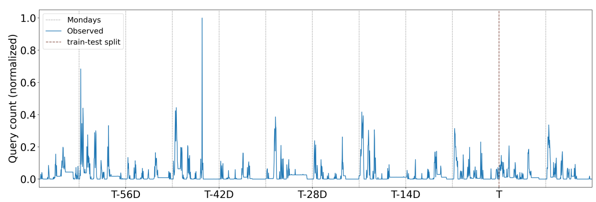
(b) Provisioned cluster P1: daily query count (QC) series



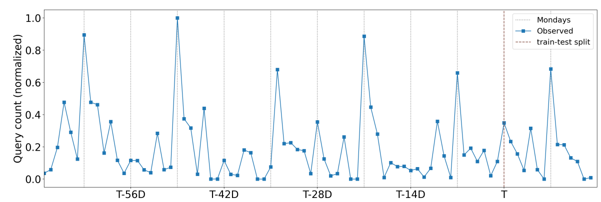
(c) Serverless cluster S2: hourly max memory (MM) series



(d) Serverless cluster S2: daily max memory (MM) series



(e) Provisioned cluster P3: hourly query count (QC) series



(f) Provisioned cluster P3: daily query count (QC) series

Figure 1: Examples of time series showing varied degrees of spikiness and seasonalities. The x-axis is the timeline, a grey vertical line marks the beginning of a new week, and the red vertical line marks the beginning of test data.

(i) *Seasonal Naive Daily*: In this method, we forecast the hourly time series of the next day using the corresponding values of its previous day. (ii) *Seasonal Naive Weekly*: In this method, we forecast the time series of the next day using the corresponding hours 7 days ago. Benchmarking analysis (which we will detail in the next section) shows that seasonal naive weekly gives better results because it is a better fit for 7 day forecast. In comparison, seasonal naive daily will replicate the observations on the last day in the training set to each of the 7 days in the forecasting horizon, which is not an effective method for long-term forecasting. Furthermore, if the daily pattern is indeed very strong (e.g., repeating every day), the seasonal naive weekly method can also give good results like the seasonal naive daily method. On the other hand, if it is the weekly pattern, but not the daily pattern, that dominates in the time series, seasonal naive daily will give poor results (e.g., the workload on Tuesday does not look at all like that on Monday), while seasonal naive weekly gives much better results. For these reasons, we use seasonal naive weekly as the default baseline in the following experiments.

Train-test Splits. For each client cluster with N weeks of data, we reserved the last two weeks for testing. In particular, we performed 2 rolling train/test splits: The first train-test split includes the first $N-2$ weeks for training and the following week for testing. Then the second train-test split includes the first $N-1$ weeks for training and the following week for testing. This represents the scenario of training a forecasting model at the end of each week and using it to forecast the query workload in the next 7 days. In our study, each train-test split is called a *test case*.

Error Metric. In this study, we use the Q-error metric [25] to capture the forecasting performance. It represents the ratio error between the true value and the forecast value, defined as follows:

$$Q\text{-}error = max\left(\frac{max(estimated, 1)}{max(true, 1)}, \frac{max(true, 1)}{max(estimated, 1)}\right) \tag{3}$$

Q-error is a symmetric metric where the best value is 1 and the worst value is infinity[2].

[2]For confidentiality reasons, we choose a relative error metric to report forecasting accuracy, instead of an absolute metric. Among relative metrics, Q-error has benefits over MAPE (mean absolute percentage error) and sMAPE (symmetric MAPE) because (a) it is symmetric regarding over-prediction versus underprediction, while MAPE favors underprediction to overprediction; (b) when the target value is close to zero, MAPE and sMAPE become extremely large, while Q-error is more mitigated.

Table 4: Q-error of the baseline for the provisioned and serverless workloads, where evaluation uses ALL hourly windows a day.

	Provisioned			Serverless		
	Query Count	Runtime Total	Bytes Scanned	Query Count	Runtime Total	Max Memory
p10	1.00	1.02	1.00	1.00	1.00	1.00
p20	1.04	1.10	1.06	1.00	1.00	1.00
p30	1.07	1.20	1.10	1.01	1.12	1.02
p40	1.14	1.28	1.17	1.08	1.25	1.12
p50	1.21	1.40	1.36	1.19	1.43	1.23
p60	1.28	1.55	1.58	1.41	1.76	1.50
p70	1.51	1.71	1.85	1.85	2.54	2.58
p80	1.78	2.20	2.50	2.98	3.91	12.91
p90	2.69	3.39	7.94	8.17	10.87	105.84

For each test case, we have 7 test days and 24 hourly windows each day. (*i*) Daily aggregation: We apply a *window selection strategy*, all windows (ALL) or top-3 high-load windows (HL) to select a set of windows per day, and report on the median error of the selected windows as the error metric of the day. (*ii*) Weekly aggregation: We then take the average of the daily error metrics across the 7 days in the test week to obtain a single error value for this test case.

If the error value of a test case is within a factor of 2, we say that this test set is "predictable". Finally, we also report the percentiles of the error metric across all the test cases of all the clusters in the provisioned/serverless dataset.

Servers: All of our experiments were run on m5.24xlarge EC2 instances, each with 96 vCPUs, 384 GB memory, and 708 GB disk space. Each server runs Amazon Linux 2 and includes time series analysis libraries (including AutoGluon version 1.1.1) running on Python 3.11.

Our analysis focuses on the following two questions:

EXPT3_3: *Are the real-world workloads predictable using the baseline method?* In this experiment, we set the window selection strategy to ALL windows (i.e., considering all the hourly windows of a day and reporting on the window of the median error, followed by the aggregation across 7 days in this test case and then aggregation across different test cases in the fleet.) Table 4 shows the baseline Q-error measurements for the 6 time series across the two datasets.

Our main observations are as follows: 1) The Q-error is kept within 1.43 for 50% of the clusters (p50 value) across all 6 time series of resource signals. 2) The Q-error increases in the p60-p90 range, while the increase is more pronounced in the serverless setting. As stated above, this is due to the fact that the serverless workloads are less regular and hence harder to predict. 3) When considering our definition of predictability being the ratio error within a factor of 2 of the ground truth, 60% to 80% of the clusters appear to be predictable (depending on the time series of choice).

EXPT3_4: *How does the performance of the baseline change for high-load windows?* In the next experiment, we change

Table 5: Q-error of the baseline for the provisioned and serverless workloads, where evaluation uses top-3 high load windows a day.

	Provisioned			Serverless		
	Query Count	Runtime Total	Bytes Scanned	Query Count	Runtime Total	Max Memory
p10	1.03	1.09	1.05	1.07	1.30	1.09
p20	1.06	1.19	1.08	1.16	1.51	1.27
p30	1.11	1.28	1.14	1.40	2.05	1.52
p40	1.20	1.46	1.27	1.86	3.01	2.39
p50	1.33	1.74	1.49	3.04	5.62	8.71
p60	1.59	2.05	1.94	5.55	12.89	65.56
p70	1.87	3.03	2.61	11.81	38.17	235.33
p80	3.04	5.50	6.57	44.45	97.67	890.32
p90	12.09	31.01	295.23	291.28	432.66	6.42k

window selection strategy to consider only the top-3 high-load windows and report on the median error of the selected windows as the error metric of the day – we make this change because forecasting use cases such as dynamic compute require accurate prediction of the peak loads of a day. Once the daily error metric is computed, we then follow the same procedure to take the average of the daily error metrics across the 7 days in this test case and aggregate error values across different test cases.

Table 5 shows the Q-error results for high-load windows. We observe a number of differences from Table 4. 1) The Q-errors increase significantly in the serverless dataset, across all the percentiles after p30 and all three time series. 2) For the provisioned dataset, the Q-errors remain relatively low in the p10-p60 range, but also increase significantly in the p80-90 range. Our profiling results show that such increases are related to the fact that when we select all the hourly windows, the median error is more likely to be reported for a window that contains a light query load. However, when we restrict our evaluation to high-load windows only, the forecasting problem focuses on peaks. This difference is especially pronounced in the serverless setting where the workload oscillates between a light load and a high peak. Such peaks are spiky in nature (not a gradual smooth pattern) and often do not follow a precise weekly pattern; instead, they may appear each week with a significant time shift, follow a bi-weekly pattern, or represent an ad-hoc burst of query load. For all of these reasons, seasonal naive weekly has difficulty forecasting peaks accurately.

3.3 Summary of Fleet Analysis Results

Revisiting our use cases, we outline the main results and implications of the fleet analysis as follows:

- For the use case of de-allocating compute resources, the forecast focuses on windows of no queries or few queries. A baseline algorithm may be relatively accurate for forecasting the windows of a light query load. Table 4 shows that the seasonal naive weekly method

can forecast light query loads within a factor of 2 for about 60%-80% of the test cases (depending on the time series of choice).
- For use cases of dynamic compute, forecasting needs to focus on high-load windows (e.g., in terms of the number of queries or concurrent memory). The patterns of these windows are intrinsically spiky, less regular in terms of the timing, and sometimes even ad-hoc. As can be seen from Table 5, the baseline method can forecast heavy query loads within a factor of 2 for about 50%-70% of the test cases in the (more regular) provisioned dataset, and 20%-40% of the test cases in the serverless dataset. For the other test cases, the errors can grow to large values. These results indicate that forecasting peak loads is a more challenging problem and requires more sophisticated algorithms.
- Serverless workloads are harder to forecast than provisioned workloads due to higher degrees of spikiness and volatility (irregularity), which also call for more advanced forecasting algorithms.

4 MODEL PERFORMANCE ANALYSIS

As we have seen before, the baseline algorithm is not sufficient for forecasting complex workloads. The salient examples are those high-load windows that exhibit spiky patterns (in contrast to a smooth growing pattern) and do not appear every week at the same time (e.g., appearing at a different time each week, appearing only every few weeks, or being an ad-hoc pattern). However, these windows represent the critical workload for resource scaling and are therefore of paramount importance. To address this issue, in this section we analyze state-of-the-art machine learning (ML) models for forecasting **high-load windows** in production workloads.

4.1 Considerations on Forecasting Models

Before delving into details of the advanced forecasting models, we first outline a number of considerations on ML-based forecasting for production clusters.

Efficiency of Model Training. Large-scale machine learning models recently gained popularity in various application domains. However, training efficiency remains our top priority for several reasons. First, following the approach of Redshift AutoWLM [30], we favor on-cluster training as it simplifies the implementation architecture as well as security-related considerations. In doing so, we expect model training to be highly efficient (in both CPU and memory usage) in order to avoid interference with the client query workload. Alternatively, models may be trained offline and such offline training requires allocating additional resources just for this purpose. Therefore, training efficiency is also important in this case in order to minimize additional resources allocated for training. Furthermore, we want to avoid relying on GPUs as they are expensive and not traditionally available in a data warehouse deployment.

Availability of Data. While many machine learning models require substantial amounts of training data, our forecasting models need to operate effectively with modest amounts of data, e.g., a couple of months of workload metadata as permitted by the data retention policy. Therefore, we seek forecasting models that can achieve optimal performance with this data availability.

Modeling Strategies. Given the above constraints, we consider a number of variable modeling strategies:

- *Instance optimized model*: We train a forecasting model for each client cluster separately. It has the potential to best fit the specific client workload, but also carries the risk of overfitting if the training dataset is not large enough. The training needs to be highly efficient as mentioned above.
- *Global model*: We train a global model across a sufficiently large set of client workloads in the offline mode. It has the potential to observe a broad set of patterns. Yet, we need to investigate if it offers optimal performance for individual client workloads.
- *Foundation model*: Another interesting option is a foundation model that has been pre-trained on large corpus of time series, promising to work out-of-the-box for predictive problems in new domains. For our resource forecasting problem, this means that no training is required for any client workloads. We can simply run the foundation model in the inference mode to forecast for a specific length, and such inference is usually efficient. We will explore whether the latest foundation models for time series forecasting can provide accurate forecasts for our problem.

"One size does not fit all." The fleet analysis in the previous section already pointed to the fact that the spikiness and seasonality patterns vary from cluster to cluster. It is unrealistic to assume that one model works well for all unless a foundation models lives up to its promise. In general, we would expect different models to suit different data characteristics (a.k.a., the inductive bias of a model). Therefore, figuring out how to select the most effective model, or a set of them, for each client workload is a critical issue.

For the above reasons, we have chosen to implement the modeling strategies (instance optimized, global, or foundation models) using **AutoGluon** [32]. It is an open-source AutoML package that provides a unified interface to forecasting models from libraries such as StatsForecast [12] or GluonTS [2]. It offers a variety of benefits for our analysis: 1) It provides a model zoo that ranges from statistical to machine learning to latest transformer models for time series forecasting. 2) It performs automated model selection to form a weighted ensemble model, which has been shown to consistently improve forecast accuracy [32]. In our case, automated model selection and ensembling are essential for finding model combinations that accurately capture the patterns unique to each specific cluster. 3) The AutoGluon model zoo also includes Chronos [3], a latest foundation model for time series forecasting.

4.2 Individual Forecasting Models

Given the aforementioned challenges in forecasting high-load windows in production workloads, our analysis in this section begins by comparing the performance of popular statistical and ML forecasting algorithms to the baseline model. Given the large set of forecasting algorithms available, including the recent wave of deep learning methods and transformers for time series data, our objective is to identify the strengths and limitations of these methods for the new task of query workload forecasting.

Models. We classify the models in the AutoGluon time series library into two categories and choose a few popular models or latest models in each category to compare to the baseline model.

Local models. The first class of forecasting algorithms are the so-called *local* models, where a separate model copy is fit for each individual time series. These include AutoARIMA [6], AutoETS [16], and Theta [4]. Furthermore, we consider two variants of the LightGBM forecaster, with and without seasonal differencing, as they are known to provide competitive performance in AutoGluon's internal analysis.

Global deep models. The second class of forecasting algorithms are *global* models. In contrast to local models, a single global model is trained on all time series simultaneously. We consider a number of popular global deep learning models. DeepAR [29] is an autoregressive forecasting model based on the LSTM architecture. We also choose two models based on the transformer architecture, PatchTST (transformer-based forecaster that segments each time series into patches) [27] and TFT (temporal fusion transformer that combines LSTM with a transformer layer) [18].

Setup. In this initial profiling study, we restrict our attention to a single metric (query count) measured across two test weeks for 10 serverless clusters that were selected using stratified sampling based on the total number of queries, resulting in 20 test cases in total. For each model, we record the following metrics:

- *End-to-end runtime* — sum of training and prediction time over all test cases;
- *Win rate vs. the baseline* — fraction of test cases where the model achieves a lower Q-error than the baseline;
- *Mean relative Q-error* [11] — geometric mean of the relative score in Q-error (model_Q-error / baseline_Q-error) across m test cases, defined as follows:

$$MRQ = \left(\prod_{i=1}^{m} \frac{Q\text{-}error^{(i)}_{\text{model}}}{Q\text{-}error^{(i)}_{\text{baseline}}} \right)^{\frac{1}{m}} \tag{4}$$

  Values less than 1 indicate that the model achieves lower error, while values above 1 mean that the model gives higher error than the baseline. Hence, the lower the value, the better the model.

In data preprocessing, we considered log transformation to turn our time series, which are generally spiky, into smoother signals. This transformation, however, makes most the models, including the best-performing models, deteriorate in accuracy, indicating too much information loss in this process. (The only exception is the foundation model to be detailed in a later study, which benefits from log transformation.) Therefore, we report accuracy results in the following experiments using the data transformation techniques most suitable for the selected models.

Hyperparameter tuning. The three deep models selected in our study have gone through extensive hyperparameter tuning. For each train-test split (test case), we tune each deep model using the validation set, which is the last two weeks of the training set, up to 100 distinct hyperparameter configurations. The model configuration that gives the best performance on the validation set is selected to run inference on the test set, for which we report the Q-error against the ground truth. Then we take the geometric mean over the Q-errors of all test cases. More specifically, we have tuned the following hyperparameters of different deep methods: (1) DeepAR: learning rate, num_layers, hidden_size, batch_size, dropout_rate, and max_epochs. (2) PatchTST: learning rate, num_encoder_layers, d_model, batch _size, max_epochs, and scaling. (2) TFT: learning rate, hidden_dim, batch_size, dropout_rate, and max_epochs.

EXPT4_1: *Can individual forecasting models provide a consistent improvement in terms of accuracy over the baseline?*

Table 6 compares the performance of each candidate model to the baseline model over 20 test cases.

Among local models, the mean relative Q-errors (MRQ) of the following four models are less than 1 (indicating them to be better than the baseline): AutoARIMA, AutoETS, LightGBM, LightGBM (weekly diff), with the last model achieving the lowest MRQ of 0.451. In contrast, SeasonalNaive (daily) and Theta lose to the baseline by the factor 2.682 and 2.797, respectively. In particular, the poor performance of SeasonalNaive (daily) is due to the fact that when the time series has a strong daily pattern, SeasonalNaive (weekly) often works similarly well, whereas when the time series has a strong

Table 6: EXPT4_1 Comparison of candidate forecasting models, both local model and global deep models, with the baseline using the QC time series of 10 serverless clusters.

Model	Local / global	Total runtime (s)	Mean relative Q-error	Win rate vs. baseline
SeasonalNaive (weekly)	local	44.8	1.000	-
SeasonalNaive (daily)	local	44.8	2.682	35%
AutoETS	local	610.8	0.941	55%
AutoARIMA	local	1104.1	0.915	50%
Theta	local	471.6	2.797	30%
LightGBM	local	36.0	0.940	40%
LightGBM (weekly diff.)	local	35.6	0.451	55%
TFT	global	8101.4	2.767	30%
DeepAR	global	78364.8	5.726	5%
PatchTST	global	6714.5	3.531	30%

weekly pattern, SeasonalNaive (daily) makes huge mistakes. For this reason, we discard SeasonalNaive (daily) and Theta due to their loss of accuracy.

Among the remaining models, we further examine their runtimes: traditional statistical models including AutoETS and AutoARIMA incur 14 times and 25 times the runtime of the baseline, respectively, while achieving only marginal improvement in accuracy, as shown by their mean relative Q-error, 0.941 and 0.915, respectively. The slow performance can be explained by the fact that the implementations of statistical models are not optimized for long seasonal periods (168 in our study) and cannot take advantage of many CPU cores. Therefore, these methods do not represent good cost-performance tradeoffs. Given our preference for high-efficient training algorithms, we further discard AutoARIMA and AutoETS in model selection. Overall, we observe LightGBM forecasters to be close to the baseline in the runtime while providing improvement in accuracy over the baseline. Therefore, we keep both LightGBM forecasters in our model selection to enable ensemble methods.

Further considering the global models, we observe that local models provide overall better runtime and forecast accuracy than the global models. At this point, we cannot conclude if this is caused by limited training data or fundamental limitations of the considered global models. Therefore, we will investigate global models in more detail in the following experiments. Among the global models, we observe TFT to provide the best accuracy in Q-error. Regarding the effect of hyperparameter tuning, TFT with hyperparameter tuning did not outperform its default configuration, which had been tuned internally by the AutoGluon developers. However, hyperparameter tuning improved the performance of DeepAR and PatchTST, in particular, by using large neural networks (more layers, more nodes in each hidden layer, or the combination of both). Despite the larger network sizes, DeepAR and PatchTST still lose to TFT in terms of accuracy. For this reason, we will keep TFT as the most competitive global model (using its default configuration) and study it further in the next set of experiments.

Finally, we observe that *none of the models provide a consistent improvement over the baseline*, as shown by the metric "win rate vs. the baseline". The candidate ML models work well for some clusters, while losing to the seasonal naive forecaster for the other clusters. Motivated by this observation, we move our attention to ensemble approaches that combine several forecasting models in an effective manner.

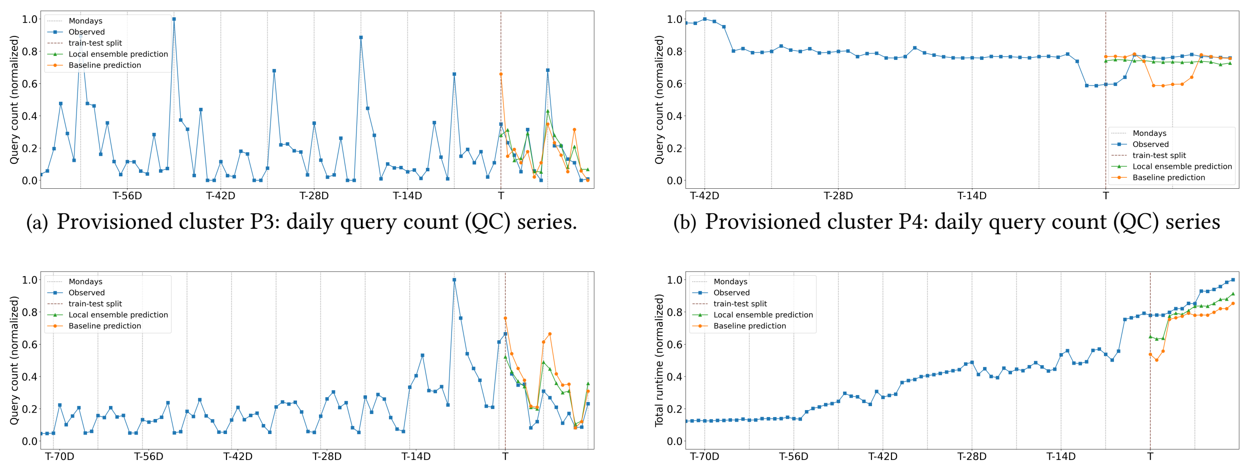
4.3 Instance Optimized and Global Ensembles

As we presented in Section 3, client clusters often exhibit diverse behaviors (e.g., varying seasonal periods or spikiness). Therefore, it is natural to expect that no single model can accurately forecast the workloads of different clusters, due to the inductive biases of individual models. Our results of the previous study validated this empirically as none of the individual models consistently outperformed the baseline.

Ensembling approaches provide one way to deal with this problem, by automatically combining several forecasting models with learned weights to indicate their contribution in the forecast [34]. AutoGluon provides an AutoML framework for constructing such ensemble models. However, the predetermined model ensembles in AutoGluon also failed on our datasets, i.e., are unable to outperform the baseline model for the majority of the clusters. Therefore, we are the first, to the best of our knowledge, to devise custom ensemble models for query workload forecasting following a systematic approach: (1) we screen through the large library of existing models and identify those that represent good time-accuracy tradeoffs, as presented in the previous study; (2) we construct ensemble models in both instance optimized and global settings, with the most suitable configuration of the training loss, known covariates (hour of the day, day of the week, day of the month), validation length and frequency. We detail these ensemble models below.

Table 7: Comparing Q-error of the AutoGluon local ensemble (LE) and global ensembles (GE1, GE2) against the baseline (B). LE and GE1 use statistical and ML models, whereas GE2 extends GE1 with Temporal Fusion Transformer (TFT).

	Provisioned												Serverless											
	Query Count (QC)				Runtime Total (RT)				Bytes Scanned (BS)				Query Count (QC)				Runtime Total (RT)				Max Memory (MM)			
	B	LE	GE1	GE2	B	LE	GE1	GE2	B	LE	GE1	GE2	B	LE	GE1	GE2	B	LE	GE1	GE2	B	LE	GE1	GE2
p10	1.03	1.04	1.04	1.05	1.09	1.09	1.11	1.12	1.05	1.05	1.07	1.11	1.07	1.09	1.12	1.35	1.30	1.27	1.34	1.45	1.09	1.12	1.18	1.61
p20	1.06	1.07	1.07	1.11	1.19	1.17	1.18	1.21	1.08	1.08	1.11	1.22	1.16	1.20	1.32	1.65	1.51	1.57	1.57	1.91	1.27	1.31	1.39	2.00
p30	1.11	1.12	1.12	1.16	1.28	1.27	1.27	1.31	1.14	1.14	1.16	1.50	1.40	1.44	1.57	2.09	2.05	1.98	2.01	2.47	1.52	1.61	1.69	2.45
p40	1.20	1.21	1.19	1.26	1.46	1.43	1.39	1.44	1.27	1.29	1.26	1.69	1.86	1.87	2.14	2.98	3.01	2.75	2.51	3.46	2.39	2.09	2.59	3.32
p50	1.33	1.35	1.34	1.41	1.74	1.68	1.64	1.67	1.49	1.46	1.46	2.00	3.04	2.74	3.06	4.05	5.62	4.28	3.74	4.61	8.71	3.50	4.69	5.56
p60	1.59	1.58	1.55	1.53	2.05	1.86	1.84	1.94	1.94	1.83	1.76	2.18	5.55	4.09	4.11	5.59	12.89	5.97	5.41	7.06	65.56	7.61	9.12	10.52
p70	1.87	1.85	1.77	1.87	3.03	2.49	2.42	2.52	2.61	2.18	2.10	2.77	11.81	5.89	6.10	9.05	38.17	9.50	7.33	11.47	235.33	26.09	16.42	21.05
p80	3.04	2.67	2.57	2.64	5.50	3.32	3.18	3.57	6.57	3.36	2.63	3.88	44.45	11.02	11.23	18.09	97.67	16.39	11.10	17.79	890.32	109.56	37.71	44.76
p90	12.09	6.73	5.41	4.68	31.01	10.22	5.14	5.41	295.23	11.06	6.00	7.12	291.28	33.87	25.90	39.79	432.66	55.34	23.47	37.90	6.42k	785.38	101.34	158.03



(a) Provisioned cluster P3: daily query count (QC) series.

(b) Provisioned cluster P4: daily query count (QC) series

(c) Serverless cluster S5: daily query count (QC) series

(d) Serverless cluster S6: daily total runtime (RT) series

Figure 2: Example time series where the local ensemble model produces a more accurate forecast than the baseline.

Ensemble Models. We start by combining the 3 most promising algorithms from the previous experiment — Seasonal-Naive (weekly) and two versions of LightGBM — in a **local ensemble** (LE). For this we use the standard weighted ensemble implementation available in AutoGluon [32]. This strategy, known as forward stepwise selection [7], defines the final forecast as a convex combination of individual model predictions. We fit a separate ensemble model for each time series. The ensemble weights are chosen by minimizing the *mean absolute error* on the validation set consisting of last two weeks of the training data.

We also consider two versions of a **global ensemble** model. Global ensemble 1 (GE1) uses the same three models as the local ensemble (LE), but shares the model weights across all clusters. The ensemble weights are learned to minimize the *average mean absolute error* on the validation set for all clusters. Another variation, global ensemble 2 (GE2) adds the Temporal Fusion Transformer (TFT) to the list of available models as it was the best performing global model in our preliminary analysis (Table 6).

EXPT4_2: *Can ensembles accurately forecast a larger fraction of workloads than the baseline model?*

To answer this question, we compare the marginal distribution of the Q-error scores achieved by all models across all cluster types and workload metrics. Table 7 reports the p10–p90 deciles of these Q-error distributions.

First, we observe that *ensemble models achieve Q-error scores comparable to the baseline in the p10-p50 range, with notable improvements occurring in the p60-p90 range.* The improvements are especially prominent on serverless workloads that are hard to predict using the baseline model. Figure 2 demonstrates some cases where the ML model produces a more accurate forecast than the baseline. In Figure 2(a), the ML model captures biweekly seasonality while the baseline models fails to do that. In particular, for the first test week, the baseline (seasonal naive weekly) copies the peak from the previous week which does not exist this week, whereas for the second test week, it misses the peak that actually exists by failing to observe the pattern from two weeks ago. In Figures 2(b)-2(c), we see that the ML model is able to quickly recover from atypical load patterns. Finally,

Table 8: Comparing the mean relative Q-error (MRQ) of our local ensemble (LE) and global ensembles (GE1, GE2) over the baseline, for different spikiness buckets of workloads. Lower values indicate more accurate forecasts compared to the baseline.

	Provisioned									Serverless								
	Query Count (QC)			Runtime Total (RT)			Bytes Scanned (BS)			Query Count (QC)			Runtime Total (RT)			Max Memory (MM)		
	LE	GE1	GE2	LE	GE1	GE2	LE	GE1	GE2	LE	GE1	GE2	LE	GE1	GE2	LE	GE1	GE2
Low Spikiness	0.89	0.86	0.88	0.74	0.70	0.72	0.67	0.57	0.66	0.54	0.53	0.73	0.45	0.35	0.52	0.33	0.23	0.32
Medium Spikiness	0.85	0.75	0.82	0.66	0.35	0.43	0.57	0.21	0.31	0.70	0.50	0.78	0.46	0.33	0.44	0.30	0.17	0.23
High Spikiness	0.93	0.85	0.90	0.87	0.68	0.61	0.34	0.22	0.28	1.04	1.30	1.20	0.70	0.61	0.65	0.65	0.43	0.56
Geometric mean	**0.88**	**0.83**	0.86	**0.74**	**0.61**	0.64	**0.61**	**0.44**	0.54	**0.62**	**0.58**	0.78	**0.48**	**0.37**	0.51	**0.35**	**0.23**	0.31

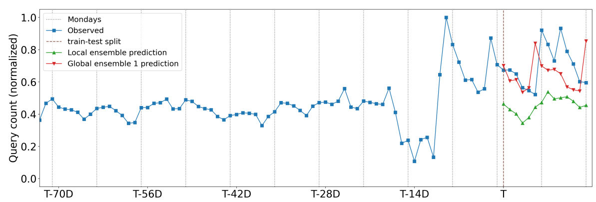


Figure 3: Serverless query count time series where the local ensemble model overfits, while global ensemble produces a reasonable forecast.

Figure 2(d) shows that the ML model provides a better fit than the baseline when a trend is present. We also note that given our data length of about 12 weeks, with the last two weeks reserved for testing, the ML models are overall not exposed to sufficient data for learning more complex monthly patterns, for which longer data history will be helpful.

Second, while local ensembles achieve more consistent improvements over a larger fraction of workloads, global ensembles (GE1) can prevent very large errors around p80-p90. The large errors for local ensemble at the tail of the distribution are caused by overfitting, i.e., when the ensemble selects models that perform poorly on the test set. Global ensemble, in contrast, shares the model weights across all clusters, which makes it more resilient to overfitting. An example of such time series is shown in Figure 3, where the local ensemble overfits the training data, which is the blue curve before the red line (the boundary of train-test split) but with the last two weeks of training data reserved for validation only.

Third, by comparing the scores of the two variants of global ensembles, GE1 and GE2 (with TFT added), we conclude that adding the Temporal Fusion Transformer (TFT) model to the ensemble (GE2) degrades the prediction accuracy. This result aligns with our preliminary conclusion that global deep learning models are not yet well-suited for workload forecasting. One possible explanation for the poor performance of TFT is the numerical instabilities caused by spiky time series (e.g., when computing the standard deviation in the LayerNorm layer), which poses a significant technical challenge to TFT.

In the above experiment we looked at the marginal distribution of errors for different models. This told us about the fraction of workloads that are predictable by each model. We now ask a different question:

EXPT4_3: *How do ensemble models compare head-to-head with the baseline for different workload types?*

We partition the time series for each resource metric into 3-bucket equi-depth histogram based on their spikiness (Equation 1). For each bucket of spikiness, we report the mean relative Q-error (MRQ) of each model, as defined in Equation 4. The results for all workloads and models are shown in Table 8. For 4 out of 6 time series, the MRQ for high spikiness time series is significantly higher compared to low and medium spikiness series. This is especially true for the time series in the serverless workload. This means that spiky time series are harder to forecast, and it is not easy to outperform the baseline in the high spikiness bucket.

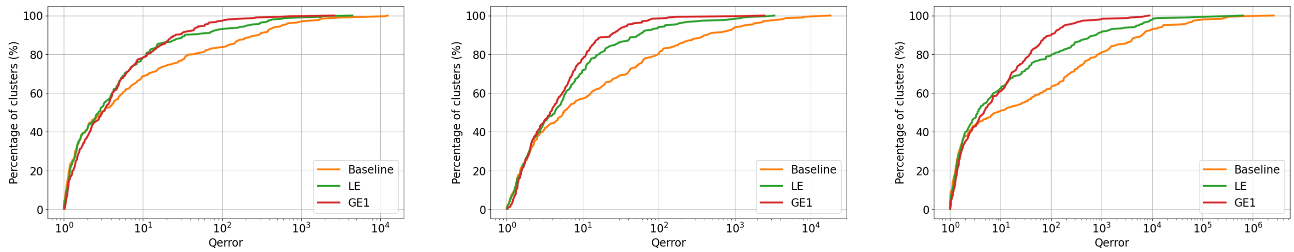
We also confirm some of the previous findings. *All ensemble models provide significant improvements over the baseline for all time series and spikiness levels (except high spikiness QC series).* Also, we again notice that GE2 is less accurate than GE1, indicating that addition of the Temporal Fusion Transformer model did not improve the accuracy. Hence, we choose GE1 as our best global ensemble, in addition to the local ensemble model (LE).

The last row of Table 8 shows the overall results of LE and GE1 across the fleet: these ensemble models *reduce 38-77% errors on average (in terms of the geometric mean of Q-errors) for the serverless workloads and 12-56% errors for the provisioned workloads*, which represent substantial reduction of errors across the fleet.

4.4 Foundation model

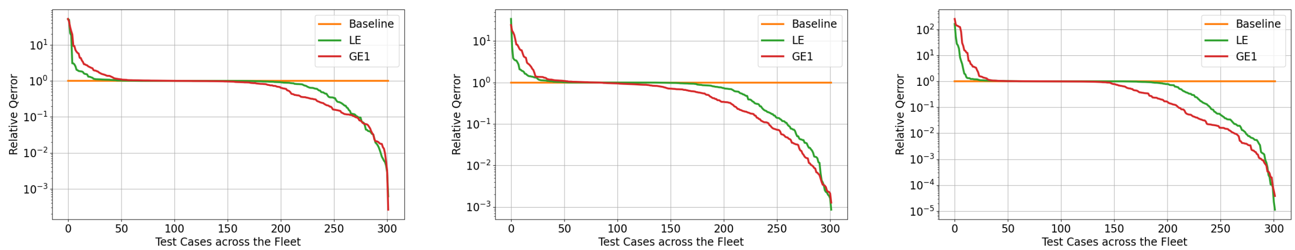
EXPT4_4: *Are pre-trained time series models more accurate than the baseline for workload forecasting?*

We next evaluate a recent time series foundation model, Chronos [3], in the context of query workload forecasting. Chronos is a transformer model pre-trained on a large corpus of real and synthetic time series data. The main advantage of Chronos is its ability to generate zero-shot forecasts on new unseen time series without requiring task-specific training.



(a) Q-error dist. of query count (QC) series (b) Q-error dist. of runtime total (RT) series (c) Q-error dist. of max memory (MM) series

Figure 4: Q-error distribution of the baseline, local ensemble (LE), and global ensemble (GE1) for the serverless workload.



(a) Relative score dist. of the QC series (b) Relative score dist. of the RT series (c) Relative score dist. of the MM series

Figure 5: Relative Q-error ($Q\text{-}error_{\textbf{model}}/Q\text{-}error_{\textbf{baseline}}$, lower is better) of local and global ensembles for the serverless workload.

We use the Chronos mini variant of the model, including 20 million parameters, as it is the largest size that provides reasonable inference speed without a GPU.

We follow the same protocol as in EXPT4_2 and compare the marginal distribution of Q-errors of Chronos to the baseline model in Table 9. We observe that the Chronos model is unable to improve over the baseline for any of the time series. The relatively poor performance of Chronos is likely caused by a combination of two factors. First, the Chronos tokenization scheme is known to lose resolution on spiky data [3, Section 5.7]. Second, the pre-training corpus of Chronos primarily contains bounded and well-behaved time series from domains like weather and electricity. These two limitations need to be addressed to make the model better suited for query workload forecasting.

Table 9: Q-error of Chronos, a foundation model for time series forecasting, with log transformation

	Provisioned			**Serverless**		
	Query Count	Runtime Total	Bytes Scanned	Query Count	Runtime Total	Max Memory
p10	1.04	1.10	1.05	1.12	1.26	1.12
p20	1.07	1.22	1.08	1.34	1.66	1.45
p30	1.17	1.37	1.19	1.89	2.39	2.09
p40	1.28	1.59	1.34	3.38	4.66	4.51
p50	1.47	1.93	1.65	7.60	12.89	33.72
p60	1.79	2.21	2.32	21.55	32.62	253.40
p70	2.19	2.90	3.16	58.14	98.79	1.07K
p80	4.20	5.28	5.76	224.17	228.04	3.55K
p90	15.98	38.27	360.57	493.36	760.78	11.27K

4.5 Validating Ensemble Models with Other Use Cases

Finally, we validate our ensemble models on other use cases to demonstrate the generality of the proposed approach.

EXPT4_5: *Ensemble models for fine-grained forecasts.* Recall from the use cases in Table 1 that intelligent resource scaling requires both long-term and short-term forecasts. In this experiment, we investigate the performance of our ensemble models for short-term forecasting, as informed by the dynamic compute use case (i.e., allocating resources dynamically for query bursts). To this end, we trained local ensemble models for 10-minute forecasts using a one-month dataset of the top 200 serverless clusters in the total number of queries. In particular, we kept the first 3 weeks as training data and created 7 rolling train-test splits: each train-test split includes the current training set and uses the next day as the test set; then the next train-test split adds the current test day into the training set and uses the following day as the test set, and so on. This simulates the retraining of the forecasting model at the end of each day, in order to adapt

Table 10: Q-error of the local ensemble (LE) model against the baseline (B) for fine-grained (10-minute) forecasts of serverless workloads

	Query Count (QC)		Runtime Total (RT)		Max Memory (MM)	
	B	LE	B	LE	B	LE
p10	1.03	1.04	1.05	1.06	1.01	1.03
p20	1.07	1.10	1.13	1.16	1.04	1.10
p30	1.13	1.16	1.25	1.29	1.10	1.20
p40	1.23	1.23	1.47	1.55	1.23	1.35
p50	1.35	1.33	2.02	1.97	1.44	1.58
p60	1.54	1.53	3.33	2.83	1.92	2.21
p70	1.89	1.88	5.89	4.27	4.47	3.63
p80	3.69	3.47	13.13	7.20	27.27	8.05
p90	15.00	9.29	52.86	16.93	227.53	35.56

more quickly to the latest trend. Other model parameters remain the same as before.

Table 10 reports on Q-errors of the baseline (B) and our local ensemble (LE) model for fine-grained (10-minute) forecasts of these serverless workloads. The results confirm our previous observations: *the local (instance optimized) ensemble model achieves Q-error scores comparable to the baseline in the p10-p50 range, and notable improvements in the p60-p90 range*, indicating its ability to reduce errors in a substantial fraction of the clusters.

EXPT4_6: *Initial evidence of end-to-end benefits.* We are in the process of developing a simulation of Redshift Serverless with intelligent AI scaling [26], which can incorporate the results and models proposed in this paper for an initial evaluation of their efficacy in a real world setting. The simulator takes the forecasts of resource signals, including the number of queries, runtime total, and maximum concurrent queries, and uses relevant heuristics to decide on the number of additionally required concurrency scaling clusters to prepare before the actual queries arrive. While the full-scale simulation of complex production behaviors and the best heuristics will be a focus of our future work, we report some initial results in this study. More specifically, we selected the top two clusters with the largest improvement in absolute error that our local ensemble offers over the baseline, and used simple heuristics to acquire additional compute resources based on the runtime total (RT) signal. Our simulation result shows that by providing a more accurate forecast of the peak loads, the local ensemble model enables the auto-scaling module to significantly optimize the internal allocation of resources, *reducing resource costs by up to 40% for the selected workloads*, while maintaining user performance metrics such as p99 and p100 query runtime (within noise levels).

4.6 Summary of Model Analysis Results

We now summarize the main results of our model analysis for forecasting high-load windows as follows:

- None of the individual machine learning models provide a consistent improvement over the straightforward baseline (seasonal naive weekly).
- Local (instance optimized) ensemble models achieve Q-error scores comparable to the baseline in the p10-p50 range of Q-error, and notable improvements in the p60-p90 range, which are especially prominent on serverless workloads. Global ensembles can prevent very large errors in the p80-p90 range of Q-error, and otherwise are comparable to or slightly worse than the local ensembles. Figure 4 summarizes the Q-error distribution of the local ensemble and the best global ensemble against the baseline for serverless time series. For the QC series, for instance, ensemble models bound the Q-error of 60% (80%) of all the test cases to 4.09-4.11 (11.02-11.23), as opposed to the baseline's Q-error of 5.55 (44.45). In general, the benefits of the ensemble models over the baseline are significant in the tail 40-50% of the test cases fleet-wide.
- Regarding the win rate, our local and best global ensemble models provide remarkable improvements over the baseline. As Figure 5 shows for the serverless workloads, ensemble models provide similar performance to the baseline for 26-30% of the test cases and wins over the baseline for 48-56% of the test cases.
- In terms of error reduction, across the fleet the ensemble models **reduce 38-77% errors on average for the serverless workloads and 12-56% errors for the provisioned workloads** (see the last row of Table 8).
- Early results from simulation experiments indicate that improvements from using the ensemble model approach can manifest promising real world benefits.
- Transformers as a global model and Chronos as a foundation model for time series forecasting are not yet effective for query workload forecasting. This is due to their limitations related to the requirement of substantially more training data or the lack of techniques to handle spiky data.

5 EXTENDED DISCUSSIONS

We now summarize the main findings of our fleet data analysis and model analysis, and highlight promising directions for future work based on these results.

No single forecasting model can provide consistent improvements over the baseline model because of diversity of the workload patterns. While low-load scenarios are served reasonably well by the baseline model, high-load windows benefit from ML-based ensemble models. These ensemble models are especially beneficial for serverless workloads that exhibit higher degrees of spikiness and volatility (irregularity) and hence are harder to forecast compared to the provisioned ones.

Model Selection and Ensembling are crucial components of the forecasting pipeline. Our model analysis results have demonstrated the superior performance of local and global ensembles that maintain the performance in the p10-p50 range of Q-error as the baseline, while mitigating the large errors in the p60-p90 range where the naive baseline suffers from the loss of accuracy.

Still, there remain important practical challenges associated with these techniques. How should training and validation data be balanced? Note here that automated model selection requires reserving the last few weeks of training data as validation data to compare models. However, we observed several cases in our study that a new trend occurred in last week (e.g., the client workload changed from a light load to a heavy load), but by reserving the last week as validation data, our ensemble model missed the opportunity to learn this trend as it was not included in the training data. Instead, should models be re-trained on the validation data after model selection is complete? It will double the training time, which may be beneficial to some clusters but wasteful for many others. Existing forecasting frameworks make these choices based on heuristics, and a more thorough theoretical or empirical investigation will be valuable to the domain of workload forecasting.

Poor Accuracy of Deep Learning Models. Existing deep learning models for time series forecasting are often designed for and evaluated on well-behaved, bounded time series from domains like electricity, traffic, or weather [9]. Query workload forecasting, which is crucial to many cloud analytics services, represents a new domain where workload metrics exhibit spiky and noisy time series, as demonstrated in this study. Such new types of time series can pose problems for deep learning architectures — for example, by causing numerical instabilities in input normalizaiton layers [17]. Since datasets used for pre-training and evaluation in deep learning literature often do not contain workload metrics data, these gaps remain unaddressed by the forecasting researchers. The database community could help bridge this gap by open-sourcing more relevant datasets and drawing attention to the unique challenges inherent to this new setting of workload metrics forecasting.

Advanced Model Development. For intelligent resource scaling, some use cases focus on light query loads, which can be well served by a baseline method, while others focus on high-load, spiky windows. For the latter category of use cases, combinations of statistical and ML models can provide reasonable forecast accuracy, within a factor of 2 in Q-error in the p60-p70 range of the (more regular) provisioned workloads, and within a factor of 3 in the p40-p50 range of serverless workloads, while mitigating large errors that naive baselines experience. Yet, making accurate forecasts (e.g., within the tight bound of factor 2-3) for high-load, spiky windows for the majority of the production clusters likely requires further innovation on model development.

Probabilistic Multivariate Forecasts. Our analysis so far has considered the use cases where it is sufficient to generate *point* forecasts of the resource metrics. However, there remain important use cases, such as determining the reserved capacity (Table 1), that would benefit from entire multivariate future usage trajectories. Existing probabilistic forecasting methods are not suitable for such tasks since they are typically limited to univariate data and only estimate the marginal (not joint) distribution of the future time series values [5]. More work is needed both on new approaches for multivariate probabilistic forecasting, as well as on addressing some fundamental challenges with evaluation of these models [22].

6 CONCLUSIONS

AI-powered intelligent resource scaling relies on the ability to forecast query workloads and their resource consumption accurately. Although the forecasting problem has been extensively studied across various domains, there is a lack of thorough analysis of existing forecasting algorithms for large-scale, real-world cloud query workloads. This paper fills this gap by providing an in-depth analysis of forecasting algorithms for real-world cloud workloads, covering the fundamental data characteristics that distinguish resource forecasting from existing forecasting problems and identifying the strengths and limitations of existing algorithms in this new domain. Our main findings point to the fact that while no single forecasting model can provide consistent improvements over the baseline model because of diversity of the workload patterns, machine learning-based ensemble models can provide remarkable reduction of forecasting errors for complex workloads that exhibit higher degrees of spikiness and volatility (irregularity). We further highlight a number of research directions that will help address the technical challenges that remain in this domain. We anticipate that our findings will provide valuable insights in informing the design of an efficient and effective solution for production use in the future, as well as in steering the forecasting community toward more effective algorithms of high real-world impact.

REFERENCES

[1] Michael Abebe, Horatiu Lazu, and Khuzaima Daudjee. 2022. Tiresias: Enabling Predictive Autonomous Storage and Indexing. *Proc. VLDB Endow.* 15, 11 (2022), 3126–3136. https://doi.org/10.14778/3551793.3551857

[2] Alexander Alexandrov, Konstantinos Benidis, Michael Bohlke-Schneider, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Danielle C Maddix, Syama Rangapuram, David Salinas, Jasper Schulz, et al. 2020. Gluonts: Probabilistic and neural time series modeling in python. *Journal of Machine Learning Research* 21, 116 (2020), 1–6.

[3] Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Syndar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, Jasper Zschiegner, Danielle C. Maddix, Hao Wang, Michael W. Mahoney, Kari Torkkola, Andrew Gordon Wilson, Michael Bohlke-Schneider, and Yuyang Wang. 2024. Chronos: Learning the Language of Time Series. *arXiv preprint arXiv:2403.07815* (2024).

[4] Vassilis Assimakopoulos and Konstantinos Nikolopoulos. 2000. The Theta model: A decomposition approach to forecasting. *International journal of forecasting* 16, 4 (2000), 521–530.

[5] Konstantinos Benidis, Syama Sundar Rangapuram, Valentin Flunkert, Yuyang Wang, Danielle Maddix, Caner Turkmen, Jan Gasthaus, Michael Bohlke-Schneider, David Salinas, Lorenzo Stella, et al. 2022. Deep learning for time series forecasting: Tutorial and literature survey. *Comput. Surveys* 55, 6 (2022), 1–36.

[6] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. 1970. *Time series analysis: forecasting and control.* John Wiley & Sons.

[7] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. 2004. Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*. 18.

[8] Chris Chatfield and Mohammad Yar. 1988. Holt-Winters forecasting: some practical issues. *Journal of the Royal Statistical Society Series D: The Statistician* 37, 2 (1988), 129–140.

[9] Zonglei Chen, Minbo Ma, Tianrui Li, Hongjun Wang, and Chongshou Li. 2023. Long sequence time-series forecasting with deep learning: A survey. *Information Fusion* 97 (2023), 101819.

[10] Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. 2023. A decoder-only foundation model for time-series forecasting. *arXiv preprint arXiv:2310.10688* (2023).

[11] Philip J Fleming and John J Wallace. 1986. How not to lie with statistics: the correct way to summarize benchmark results. *Commun. ACM* 29, 3 (1986), 218–221.

[12] Federico Garza, Max Mergenthaler Canseco, Cristian Challu, and Kin G. Olivares. 2022. StatsForecast: Lightning fast forecasting with statistical and econometric models. https://github.com/Nixtla/statsforecast.

[13] Xiao He, Ye Li, Jian Tan, Bin Wu, and Feifei Li. 2023. OneShotSTL: One-Shot Seasonal-Trend Decomposition For Online Time Series Anomaly Detection And Forecasting. *Proc. VLDB Endow.* 16, 6 (2023), 1399–1412. https://doi.org/10.14778/3583140.3583155

[14] Hanxian Huang, Tarique Siddiqui, Rana Alotaibi, Carlo Curino, Jyoti Leeka, Alekh Jindal, Jishen Zhao, Jesús Camacho-Rodríguez, and Yuanyuan Tian. 2024. Sibyl: Forecasting Time-Evolving Query Workloads. *Proc. ACM Manag. Data* 2, 1 (2024), 53:1–53:27. https://doi.org/10.1145/3639308

[15] R.J. Hyndman and G. Athanasopoulos. 2021. *Forecasting: Principles and Practice.* OTexts. https://books.google.de/books?id=gZB-zgEACAAJ

[16] Robin John Hyndman, Anne B Koehler, J Keith Ord, and Ralph David Snyder. 2008. *Forecasting with Exponential Smoothing: The State Space Approach.* Springer.

[17] Taesung Kim, Jinhee Kim, Yunwon Tae, Cheonbok Park, Jang-Ho Choi, and Jaegul Choo. 2021. Reversible instance normalization for accurate time-series forecasting against distribution shift. In *International Conference on Learning Representations*.

[18] Bryan Lim, Sercan Ö Arık, Nicolas Loeff, and Tomas Pfister. 2021. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting* 37, 4 (2021), 1748–1764.

[19] Lin Ma, Dana Van Aken, Ahmed Hefny, Gustavo Mezerhane, Andrew Pavlo, and Geoffrey J. Gordon. 2018. Query-based Workload Forecasting for Self-Driving Database Management Systems. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein (Eds.). ACM, 631–645. https://doi.org/10.1145/3183713.3196908

[20] 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. *Proc. ACM Meas. Anal. Comput. Syst.* 6, 2, Article 26 (June 2022), 28 pages. https://doi.org/10.1145/3530892

[21] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. 2018. The M4 Competition: Results, findings, conclusion and way forward. *International Journal of forecasting* 34, 4 (2018), 802–808.

[22] Étienne Marcotte, Valentina Zantedeschi, Alexandre Drouin, and Nicolas Chapados. 2023. Regions of reliability in the evaluation of multivariate probabilistic forecasts. In *International Conference on Machine Learning*. PMLR, 23958–24004.

[23] Venkata Vamsikrishna Meduri, Kanchan Chowdhury, and Mohamed Sarwat. 2021. Evaluation of Machine Learning Algorithms in Predicting the Next SQL Query from the Future. *ACM Trans. Database Syst.* 46, 1 (2021), 4:1–4:46. https://doi.org/10.1145/3442338

[24] Abhinav Mishra, Ram Sriharsha, and Sichen Zhong. 2022. OnlineSTL: Scaling Time Series Decomposition by 100x. *Proc. VLDB Endow.* 15, 7 (2022), 1417–1425. https://doi.org/10.14778/3523210.3523219

[25] Guido Moerkotte, Thomas Neumann, and Gabriele Steidl. 2009. Preventing Bad Plans by Bounding the Impact of Cardinality Estimation Errors. *Proc. VLDB Endow.* 2, 1 (2009), 982–993. https://doi.org/10.14778/1687627.1687738

[26] Vikram Nathan, Vikramank Y. Singh, Zhengchun Liu, Mohammad Rahman, Andreas Kipf, Dominik Horn, Davide Pagano, Gaurav Saxena, Balakrishnan Narayanaswamy, and Tim Kraska. 2024. Intelligent Scaling in Amazon Redshift. In *Companion of the 2024 International Conference on Management of Data, SIGMOD/PODS 2024, Santiago AA, Chile, June 9-15, 2024*, Pablo Barceló, Nayat Sánchez Pi, Alexandra Meliou, and S. Sudarshan (Eds.). ACM, 269–279. https://doi.org/10.1145/3626246.3653394

[27] Yuqi Nie, Nam H Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. 2023. A time series is worth 64 words: Long-term forecasting with transformers. *International Conference on Learning Representations* (2023).

[28] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd C. Mowry, Matthew Perron, Ian Quah, Siddharth Santurkar, Anthony Tomasic, Skye Toor, Dana Van Aken, Ziqi Wang, Yingjun Wu, Ran Xian, and Tieying Zhang. 2017. Self-Driving Database Management Systems. In *8th Biennial Conference on Innovative Data Systems Research, CIDR 2017, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2017/papers/p42-pavlo-cidr17.pdf

[29] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. 2020. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting* 36, 3 (2020), 1181–1191. https://ideas.repec.org/a/eee/intfor/v36y2020i3p1181-1191.html

[30] Gaurav Saxena, Mohammad Arifur Rahman, Naresh Chainani, Chunbin Lin, George Caragea, Fahim Chowdhury, Ryan Marcus, Tim Kraska, Ippokratis Pandis, and Balakrishnan (Murali) Narayanaswamy. 2023. Auto-WLM: Machine learning enhanced workload management in Amazon Redshift. In *SIGMOD/PODS 2023*. https://www.amazon.science/publications/auto-wlm-machine-learning-enhanced-workload-management-in-amazon-redshift

[31] 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 Conference on Usenix Annual Technical Conference (USENIX ATC'20)*. USENIX Association, USA, Article 14, 14 pages.

[32] Oleksandr Shchur, Ali Caner Turkmen, Nick Erickson, Huibin Shen, Alexander Shirkov, Tony Hu, and Bernie Wang. 2023. AutoGluon–TimeSeries: AutoML for probabilistic time series forecasting. In *International Conference on Automated Machine Learning*. PMLR.

[33] Alexander van Renen, Dominik Horn, Pascal Pfeil, Kapil Eknath Vaidya, Wenjian Dong, Murali Narayanaswamy, Zhengchun Liu, Gaurav Saxena, Andreas Kipf, and Tim Kraska. 2024. Why TPC is not enough: An analysis of the Amazon Redshift fleet. In *VLDB 2024*. https://www.amazon.science/publications/why-tpc-is-not-enough-an-analysis-of-the-amazon-redshift-fleet

[34] Xiaoqian Wang, Rob J Hyndman, Feng Li, and Yanfei Kang. 2023. Forecast combinations: An over 50-year review. *International Journal of Forecasting* 39, 4 (2023), 1518–1547.

[35] Qingsong Wen, Jingkun Gao, Xiaomin Song, Liang Sun, Huan Xu, and Shenghuo Zhu. 2018. RobustSTL: A Robust Seasonal-Trend Decomposition Algorithm for Long Time Series. *CoRR* abs/1812.01767 (2018). arXiv:1812.01767 http://arxiv.org/abs/1812.01767

[36] Qingsong Wen, Kai He, Liang Sun, Yingying Zhang, Min Ke, and Huan Xu. 2020. RobustPeriod: Time-Frequency Mining for Robust Multiple Periodicities Detection. *CoRR* abs/2002.09535 (2020). arXiv:2002.09535 https://arxiv.org/abs/2002.09535

[37] Qingsong Wen, Zhe Zhang, Yan Li, and Liang Sun. 2020. Fast RobustSTL: Efficient and Robust Seasonal-Trend Decomposition for Time Series with Complex Patterns. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash (Eds.). ACM, 2203–2213. https://doi.org/10.1145/3394486.3403271

[38] Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka. 2017. A multi-horizon quantile recurrent forecaster. *arXiv preprint arXiv:1711.11053* (2017).

[39] Gerald Woo, Chenghao Liu, Akshat Kumar, Caiming Xiong, Silvio Savarese, and Doyen Sahoo. 2024. Unified training of universal time series forecasting transformers. *International Conference on Machine Learning (ICML)* (2024).