Predicting the Costs of Serverless Workflows

Simon Eismann
University of Würzburg
Würzburg, Germany
simon.eismann@uni-wuerzburg.de

Johannes Grohmann
University of Würzburg
Würzburg, Germany
johannes.grohmann@uni-wuerzburg.de

Erwin van Eyk
Vrije Universiteit
Amsterdam, Netherlands
E.vanEyk@atlarge-research.com

Nikolas Herbst
University of Würzburg
Würzburg, Germany
nikolas.herbst@uni-wuerzburg.de

Samuel Kounev
University of Würzburg
Würzburg, Germany
samuel.kounev@uni-wuerzburg.de

ABSTRACT

Function-as-a-Service (FaaS) platforms enable users to run arbitrary functions without being concerned about operational issues, while only paying for the consumed resources. Individual functions are often composed into workflows for complex tasks. However, the pay-per-use model and nontransparent reporting by cloud providers make it challenging to estimate the expected cost of a workflow, which prevents informed business decisions. Existing cost-estimation approaches assume a static response time for the serverless functions, without taking input parameters into account.

In this paper, we propose a methodology for the cost prediction of serverless workflows consisting of input-parameter sensitive function models and a monte-carlo simulation of an abstract workflow model. Our approach enables workflow designers to predict, compare, and optimize the expected costs and performance of a planned workflow, which currently requires time-intensive experimentation. In our evaluation, we show that our approach can predict the response time and output parameters of a function based on its input parameters with an accuracy of 96.1%. In a case study with two audio-processing workflows, our approach predicts the costs of the two workflows with an accuracy of 96.2%.

CCS CONCEPTS

• **Software and its engineering → Software performance**; • **Computer systems organization → Cloud computing**; • **Computing methodologies → Modeling and simulation**; **Machine learning**.

KEYWORDS

Serverless, Workflows, Prediction, Cost, Performance

ACM Reference Format:
Simon Eismann, Johannes Grohmann, Erwin van Eyk, Nikolas Herbst, and Samuel Kounev. 2020. Predicting the Costs of Serverless Workflows. In *Proceedings of the 2020 ACM/SPEC International Conference on Performance Engineering (ICPE '20), April 20–24, 2020, Edmonton, AB, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3358960.3379133

ICPE '20, April 20–24, 2020, Edmonton, AB, Canada

ACM ISBN 978-1-4503-6991-6/20/04…$15.00
https://doi.org/10.1145/3358960.3379133

1 INTRODUCTION

Serverless computing is an emerging cloud computing paradigm, where all operational concerns, such as deployment and resource provisioning, are delegated to the cloud platform and costs are calculated based on a pay-per-use basis [19, 28]. Function-as-a-Service (FaaS) platforms, such as AWS Lambda, Google Cloud Functions, or Azure Functions enable the serverless execution of stateless, ephemeral compute functions [25, 33]. To implement complex business functionality, individual functions can be composed into *serverless workflows* using e.g. AWS Step Functions, Google Cloud Composer, or Azure Durable Functions [37].

Currently, all major cloud providers use the same cost model for serverless functions, where the cost of a function execution depends on: i) the response time of a function rounded up to the nearest 100 ms, ii) the memory allocated to the function and iii) a static charge for every invocation [2]. While many organizations report significant cost savings by switching from traditional hosting options to serverless solutions [2, 4, 26], an inhibiting factor for the adoption of serverless solutions in practice is the difficulty of estimating the expected costs of serverless functions and workflows [2, 6, 38]. A reason for this is that, in contrast to traditional hosting options, the cost of a function depends directly on its input parameters—since the response time distribution of a function depends on its input parameters. For example, the time required to resize an image depends on its original size. Therefore, the cost of resizing an image depends on its original size as well. This is exacerbated in workflows, where function outputs are often propagated to succeeding functions. Hence, the cost and response time of functions contained within a workflow can be erratic, which makes predicting the cost for the overall serverless workflow challenging.

Existing approaches for the cost estimation of serverless functions and workflows require the user to estimate the function response time with a single mean value [13, 18]. This is often inaccurate due to the erratic response time of functions within a workflow. The response time of a function or workflow can be measured using micro-benchmarks [6]. However, measuring cost is cumbersome as cost/usage statistics for serverless functions are usually delayed by 4-48 hours and aggregated either per hour or per day. Queuing

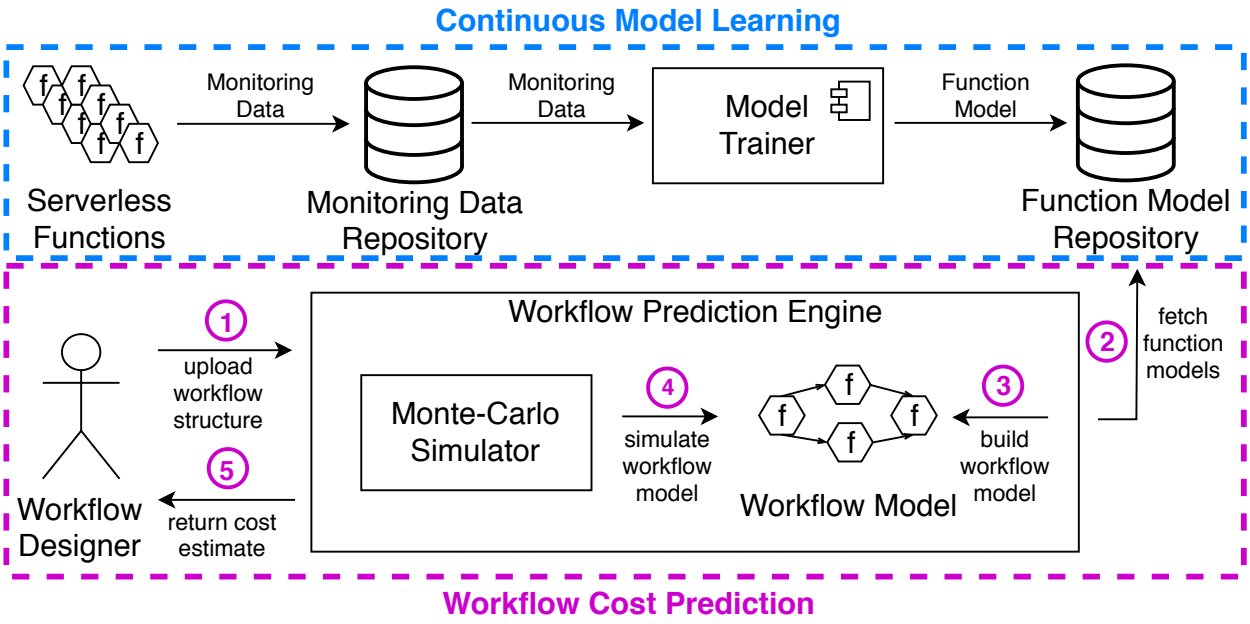


Figure 1: Overview of the proposed approach for the cost prediction of serverless workflows.

theory-based models can predict the impact of input parameters on the performance of traditional systems [1, 10, 16], but are inapplicable for serverless solutions as they require knowledge about the underlying resource landscape and deployment.

In this paper, we propose a methodology for the cost prediction of serverless workflows. First, we apply machine learning to predict the response time and output parameter distributions for individual serverless functions. Standard regression techniques, such as SVR, MARS, or random forest, can only be used to predict the mean response time of a function. However, accurate cost estimations require the prediction of response time distributions, because cloud providers round the billed execution time up to the nearest 100 ms. Therefore, we show how Mixture Density Networks (MDNs) can be used to accurately predict the response time and output parameter distributions of serverless functions. These individual function models are composed to a workflow model that describes the parameter relationships within the workflow. Finally, a Monte-Carlo simulation traverses the workflow model and samples distributions from the individual functions models to derive cost predictions for serverless workflows. In our case study, the proposed approach predicts the response time distribution and the distribution of the output parameters of five representative Google Cloud Functions with a mean accuracy of 96.1%. For two workflows composed of these functions, our approach achieves a mean workflow cost prediction accuracy of 96.2%.

The approach presented in this paper provides accurate cost predictions for previously unobserved serverless workflows. Using our approach, solution architects can make informed decisions when choosing between a serverless workflow and a traditionally hosted workflow by providing concrete numbers for the costs of the serverless workflow. Based on our cost predictions, workflow designers can compare alternatives without time-intensive experimentation. Additionally, our approach represents a first step towards fully automated workflow optimization using multi-objective optimization techniques, analogously to existing tools for traditional software systems [3, 35].

2 APPROACH

In this paper, we propose an approach to predict the costs of serverless workflow executions. Section 2.1 gives an overview of the approach, whereas Section 2.2 goes into detail on predicting the distributions of the response time and output parameters of a serverless function. Section 2.3 describes how the proposed Monte-Carlo simulation uses the predictions for individual functions to estimate the average cost per execution of a serverless workflow.

2.1 Overview

The proposed approach shown in Figure 1 can be separated into two phases, the continuous model learning process and the workflow cost prediction process.

During the continuous model learning process, the existing `Serverless Functions` are monitored. For any functions that are not already deployed in production, micro-benchmarks can be used to generate monitoring data [6]. The resulting monitoring data is stored in a `Monitoring Data Repository` (e.g., Prometheus, InfluxDB or a managed monitoring solution from the cloud provider). Periodically, the `Model Trainer` is triggered to train models that describe the response time and output parameter distributions of the serverless functions based on their input parameters, which is discussed in detail in Section 2.2. As the `Model Trainer` could make use of GPU-based acceleration during the model learning, it could be deployed in a distributed data analytics cluster with GPU acceleration, such as a Spark or Hadoop cluster.

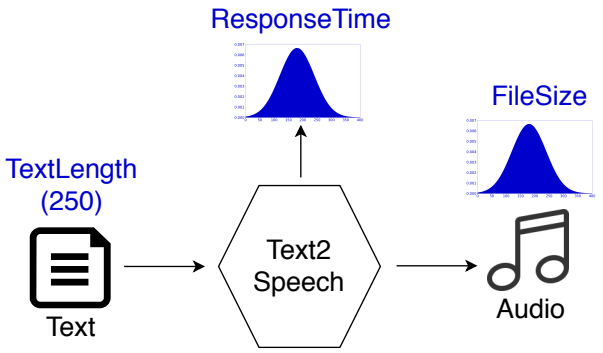


Figure 2: For multiple text segments of length 250, a distribution of response times and output file sizes can be observed for a function that transcribes text into speech.

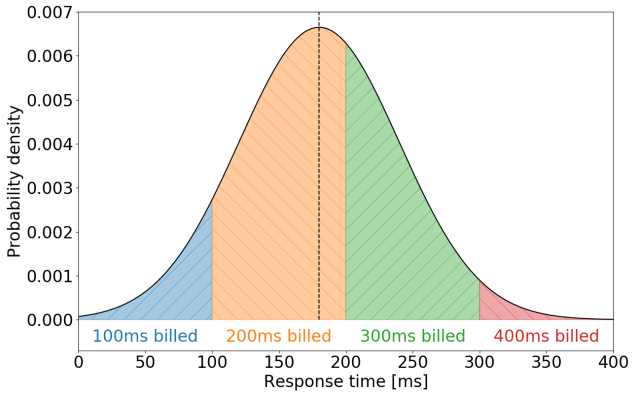


Figure 3: Comparison between billed response time and mean response time of normal distribution.

The resulting models are then stored in the `Function Model Repository`, which due to the infrequent access pattern can be a cloud data storage, such as Amazon S3, Google Cloud Storage, or Azure Storage.

The workflow cost prediction process is triggered when a workflow designer uploads the workflow that he/she wants to evaluate. Next, the `Workflow Prediction Engine` fetches the models for all functions contained in the workflow from the `Function Model Repository`. These function models are then composed into a `Workflow Model` based on the structure uploaded by the workflow designer. To derive cost predictions from the `Workflow Model`, the `Monte-Carlo Simulator` simulates the `Workflow Model`. Finally, the derived cost estimates are returned to the workflow designer. The `Workflow Prediction Engine` could be implemented as a serverless function, as it has an infrequent, potentially bursty access pattern and model inference rarely relies on GPU acceleration [43], which is currently not supported for serverless functions [24].

In the following Sections 2.2 and 2.3, we describe the prediction of function response time and output parameter distributions and our approach to derive cost estimates for serverless workflows based on the individual function models.

2.2 Function response time and output parameter distribution prediction

We train an individual model for the response time and for each output parameter of a serverless function based on monitoring data from the `Monitoring Data Repository`. This monitoring data contains the response time and parameterization for each request to the serverless function. Most machine learning techniques require numeric input, while the parameters of a function call are not necessary numeric values. Examples of non-numeric values include strings, lists, binary data, etc. In this paper, we do not address the task of creating numeric features based on this data, as there is extensive prior work targeting the automated extraction of numeric features based on function input parameters [22, 31].

For the repeated execution of a serverless function with identical input parameter characteristics, a distribution of response times and output parameters can be observed. To illustrate this, we implemented and evaluated a function called *Text2Speech*, which transcribes text segments to speech, as shown in Figure 2. Transcribing multiple text segments with a length of 250 characters, we observe a distribution of the response time due to variation in the performance and saturation of the hardware executing the function. Additionally, we also observe varying values for the size of the resulting audio file. However, both the response time and the resulting file size are closely correlated to the length of the transcribed text segment.

Predicting the distribution of the response time of a serverless function is important to estimate the resulting costs. Predicting only the expected mean response time can lead to inaccurate cost predictions, as all major FaaS providers round the billed response time up to the nearest 100 ms. Figure 3 shows this for a simulated serverless function with a normally distributed response time with a mean of 180 ms and a standard deviation of 60 ms. If we would solely use the mean response time of 180 ms and round to the nearest 100 ms, we would predict that an execution of this function is billed for 200 ms on average. However, looking at the actual probabilities of being billed 100 ms (9.12%), 200 ms (53.93%), 300 ms (34.67%) and 400 ms (2.28%), results in a mean billed time of 230.11 ms. Therefore, accurate cost estimations for serverless functions and workflows require predicting the response time distribution instead of only the mean response time.

Common regression techniques, such as SVR, MARS, or random forest can only be used to predict the mean response time of a serverless function. Therefore, we propose the usage of so-called mixture density networks (MDNs) [9]. Bishop et al. propose the idea to use a dense neural network to parameterize a gaussian mixture model. A mixture model describes the probability density function of a random variable as a linear combination of m gaussian kernels:

$$p(y|x) = \sum_{i=1}^{m} \alpha_i(x) * \phi_i(y|x) \quad (1)$$

with α_i as the mixing factor ($\sum_{i=1}^{m} \alpha_i(x) = 1$) and $\phi_i(y|x)$ as a gaussian kernel with mean μ_i and standard deviation σ_i. Provided with a large enough number of kernels, a gaussian mixture distribution

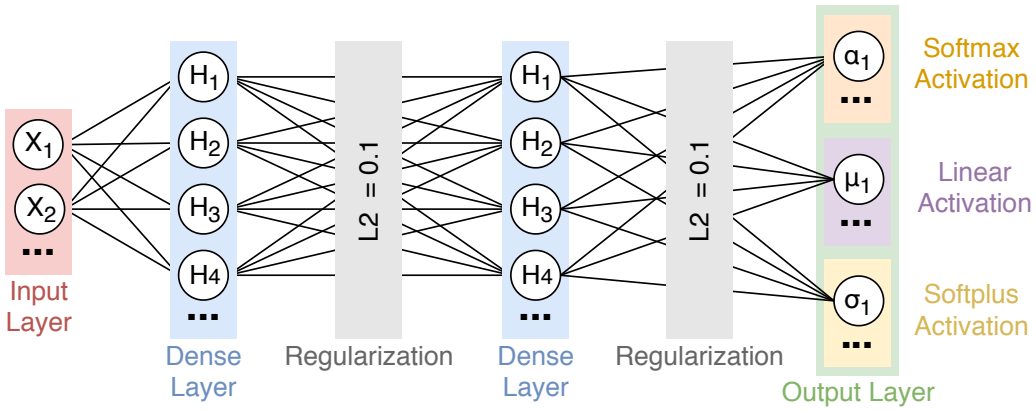


Figure 4: Mixture density network architecture for the prediction of response time and output parameter distributions of serverless functions with α as mixing coefficients, μ as kernel means and σ as kernel standard deviations.

can approximate any probability distribution with an arbitrary accuracy [20]. Simply put, a gaussian mixture distribution is the weighted average of m normal distributions. Parameterizing a gaussian mixture distribution requires the weights, means and standard deviations of the m normal distributions. In a mixture density network, these parameters are estimated using a dense neural network. As the weights, means and standard deviations are not contained within the training data set, traditional loss functions for regression, such as Mean Squared Error (MSE), Mean Squared Logarithmic Error (MSLE) or Mean Absolute Error (MAE) cannot be applied. Instead, most mixture density networks use the negative log-likelihood function as a loss function, which is defined as:

$$\ell(x) = -log(p(y|x)) \tag{2}$$

For each sample in a training batch, the logarithm of its occurrence likelihood is calculated and then negated, as neural network optimizers aim to minimize the loss function.

Figure 4 shows the network layout we propose to use for the prediction of the response time and output parameter distributions of serverless functions. It consists of an input layer, two dense hidden layers, two regularizations and an output layer that aggregates over the three layers describing the mixing coefficients α_i, the means μ_i and the standard deviations σ_i of the gaussian kernels. The input layer contains a neuron for each input parameter, so the overall network has rather few input neurons. The output layer has a total of $m * 3$ neurons, but our evaluation showed that the prediction of the response time and output parameter distributions requires usually less than five kernels. Therefore, the total number of output neurons remains usually below fifteen. With limited input and output neurons, the dense layers require only a comparatively low number of neurons (200 were sufficient during our case study).

Some input parameters have a large range of values, such as the file size. For such input parameters it is possible to only have a single observation for a specific input parameter value. Additionally, the response time for serverless functions is prone to outliers due to function cold starts [7]. If such an outlier is the only sample for its input parameter value, the neural network will overfit by parameterizing the mixture distribution for this specific input parameter value much larger than for adjacent input parameter values. In order to prevent this type of overfitting, we apply L2 regularization after each dense layer. A L2 regularization (also known as ridge regularization or Tikhonov regularization) adjusts the cost function for the gradient descent learning by adding the squared Euclidean norm of the corresponding layers weight matrix [15]. Therefore, the L2 regularization penalizes model complexity. In our use case, this is a desirable property as we assume that the relationship between an input parameter and the observed response time distribution is roughly continuous.

The dense layers use the widespread rectified linear unit (relu) activation function [32]. The output layer for the mixing coefficients uses the softmax activation function to guarantee that the mixing coefficients sum up to one. As no restrictions apply for the means of the linear kernels, the corresponding output layer uses a linear activation function. For the prediction of response time distributions, it could be restricted to positive values. However, there might be edge cases in which the distribution of an output parameter might contain negative values. As a standard deviation is restricted to values greater than or equal to zero, the corresponding output layer should also be restricted accordingly as otherwise the negative loss likelihood can no longer be calculated. Bishop et al. originally proposed the usage of an exponential activation function [9]. However, this is reported to potentially lead to numerical instability [11]. As alternatives, we tested the softplus activation function [21] and an exponential linear unit (elu) activation function [14] with an offset of 1. The convex nature of the softplus activation function enabled the network to fit linear kernels with a small standard deviation, whereas the elu + 1 activation function consistently skewed towards kernels with large standard deviations. Linear kernels with a small standard deviation are useful in mixture density networks to explain subpopulations. Therefore, we choose the softplus activation function for the standard deviation output layer. Finally, the three output layers are concatenated to form a single output layer.

2.3 Workflow cost prediction

In Section 2.2, we propose the usage of mixture density networks to predict the response time and output parameter distribution of

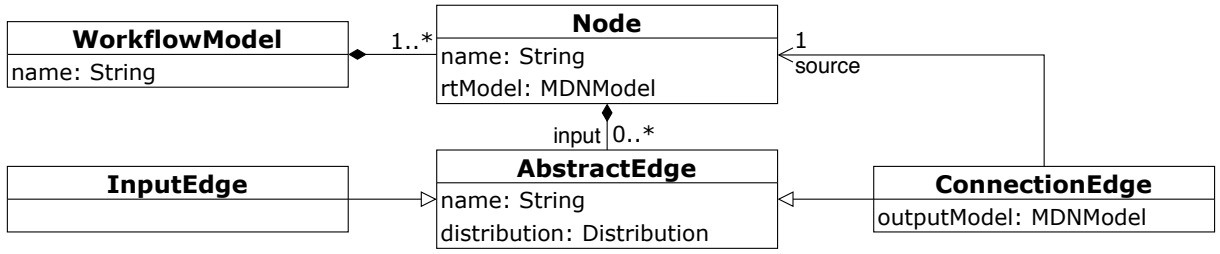


Figure 5: Meta-model for the workflow model.

Algorithm 1 Workflow Model Traversal

```
1:  function ESTIMATECOSTS(workflowModel)
2:      for edge in workflowModel.nodes.input do
3:          SOLVE(edge)
4:      end for
5:
6:      workflowCost = 0
7:      for node in workflowModel.nodes do
8:          rDist = SIMULATE(node.rtModel, node.input)
9:          functionCost = ESTIMATECOST(rDist)
10:         workflowCost += functionCost
11:     end for
12:
13:     return workflowCost
14: end function
15:
16: function SOLVE(edge)
17:     if edge.distribution != NULL then
18:         return
19:     end if
20:
21:     inputDists = edge.source.input
22:     for dependency in inputDists do
23:         SOLVE(dependency)
24:     end for
25:
26:     edge.distribution=SIMULATE(edge.model, inputDists)
27: end function
```

Algorithm 2 Monte-Carlo Simulation

```
1:  function SIMULATE(MDNModel, paramDists)
2:      numSamples = 5000
3:      resultDistList = new List()
4:      for i = 1; i ≤ numSamples; i++ do
5:          params = new List()
6:          for param in paramDists do
7:              sample = param.drawSample()
8:              params.add(sample)
9:          end for
10:         dist = MDNModel.predict(params)
11:         resultDistList.add(dist)
12:     end for
13:
14:     return new MixtureDistribution(resultDistList)
15: end function
```

individual serverless function for a concrete input parameter value. However, in a serverless workflow the input parameters of each function are a distribution instead of a concrete value, because they are the output of previous functions. We propose to run a Monte-Carlo simulation [41] on a model of the workflow to empirically determine the response time and output parameter distribution for a given distribution of input parameters.

As shown in Figure 5, the proposed workflow model is an extended, directed acyclic graph (DAG), a common formalism to model workflows [40]. As a simplification, we assume that control and data flow are identical. Each `WorkflowModel` consists of a number of `Nodes`. A `Node` represents a single execution of a serverless function and should be named after the function it represents. Every `Node` contains a number of `AbstractEdges`, which represent the input parameters to the function and also should be named accordingly. Each edge describes the name of a parameter and its corresponding distribution. There are two sub-classes of `AbstractEdge`, namely `InputEdge` and `ConnectionEdge`. An `InputEdge` represents an input to the workflow and characterizes the distribution of an input parameter to the first `Node`s in the workflow. On the contrary, `ConnectionEdge`s serve both as input parameters to nodes and output parameter from nodes. They characterize the output distribution of a return parameter of a node, which is usually an input parameter to another `Node` in the workflow. Therefore, `ConnectionEdge`s contain an `MDNModel` that can predict the `Distribution` of the output parameter the `ConnectionEdge` represents, based on the input parameters of the corresponding `Node`. `ConnectionEdge`s additionally reference the `Node` of which the output parameter originated from. Similarly, each `Node` contains a `MDNModel` that can be used to estimate the response time distribution of the serverless function represented by the `Node` based on its input parameters. Since all edges always describe input parameters of nodes, it is possible to add output parameters that do not impact the cost of the workflow execution.

Algorithm 1 requires a `WorkflowModel` as an input and provides an estimation of the costs for executing this workflow. First, at line 2-4 of Algorithm 1, all input distributions are solved by iterating over all input edges of all nodes of the workflow and recursively solving them. The `SOLVE` function described at line 16-27 returns at line 18, if a given edge already has a distribution. This is the case for `InputEdge`s for example, as they are already parameterized as

Algorithm 3 Cost Estimation

```
1:  function ESTIMATECOST(respDist)
2:      numSamples = 5000
3:      sumCosts = 0
4:      for i = 1; i ≤ numSamples; i++ do
5:          sample = respDist.drawSample()
6:          billedIntervals = CEIL(sample/BILLINGINTERVAL)
7:          sumCosts += billedIntervals * CPUCOST
8:          sumCosts += billedIntervals * MEMORYCOST
9:          sumCosts += EXECUTIONCOST
10:     end for
11:
12:     return sumCosts/numSamples
13: end function
```

input. However, if the distribution of an edge is unknown, the distribution of the edge can be estimated by applying the Monte-Carlo simulation on the given MDN model of the edge and the distributions of its input parameters. However, as the input parameters might also be unknown, all dependent edges are first solved by recursively calling SOLVE on them. This recursion is guaranteed to be finite, as DAGs are not allowed to contain circles and all input edges are already parameterized.

After the recursion ends at line 5 of Algorithm 1, all input distributions to all nodes in the given workflow model are known. Hence, lines 6-11 iterate over all nodes in the workflow engine, and sums up the estimated cost for each predicted response time distribution. Finally, the total costs can be returned at line 13.

Algorithm 2 details how the Monte-Carlo simulation derives the distribution of response times or output parameters of a serverless function based on the distribution of its input parameters. It uses the mixture density networks described in Section 2.2, that predict the expected distribution for concrete input parameter values. In lines 5-9 of Algorithm 2, the algorithm draws a sample from the probability distribution of each input parameter. Next, at lines 10-11, the mixture density model is used to predict the expected distribution for this set of input parameters and the resulting distribution is added to a list. The more samples are used in a Monte-Carlo simulation, the more precise the resulting estimation becomes, at the cost of increased computation time. As rare events are not expected in our use case, 5,000 samples likely provide a sufficient prediction accuracy. The resulting list of probability distributions is then composed to a mixture distribution with equal weights, which can be seen as the average over the individual distributions.

Algorithm 3 shows the adapted Monte-Carlo simulation used to predict the average cost for a function estimation based on its response time distribution. First, 5,000 samples are drawn from the response time distribution of the serverless function. In line 6, the algorithm calculates the number of billed intervals by dividing the response time sample by the size for the billing interval and rounding up. The number of billed intervals is then used for calculating the cost for the CPU time and the memory time, by multiplying them at lines 7 and 8 of Algorithm 3. Some cloud providers do not split the costs of CPU time and memory time; in this case lines 7 and 8 can be concatenated and replaced by just one multiplication with the charged amount per interval. Additionally, each sample is charged a constant blanket fee per execution, a.k.a. the invocation cost for each function execution. After calculating all samples, the costs for each sample are summed up and divided by the number of samples to determine the average execution cost at line 12. The static variables `BILLINGINTERVAL`, `CPUCOST`, `MEMORYCOST` and `EXECUTIONCOST` depend on the pricing of the selected cloud provider and can be parameterized accordingly.

3 CASE STUDY

We design our case study in order to answer the following three research questions:

- **RQ1:** Are mixture density networks capable of accurately predicting the distribution of the response time and the output parameters of a serverless function?
- **RQ2:** Can the proposed algorithm and the underlying machine learning models for the individual functions accurately predict the costs of a previously unobserved workflow?
- **RQ3:** What is the required time for training and workflow prediction? Is the overhead feasible for a production environment?

Based on these research questions, we implemented the following five audio utility functions on Google Cloud Functions using Python with the following input and output parameters:

Text2Speech Transcribes text files into audio files, a functionality that is commonly used to increase accessibility, to automate phone banking or in smart home assistants like Alexa or Google Assistant. It uses the google text-to-speech Python library `gTTS` (v2.0.3), which returns an MP3 file.

Parameters:

- **[Input]** *TextLength:* Length of the text that needs to be transcribed, measured in number of characters.
- **[Output]** *FileSize:* Size of the resulting MP3 file in bytes.

ProfanityDet Detects racial slurs, sexually explicit language and general expletives in a text segment. The implementation is based on the Python library `profanity` (v1.1.0), which implements a blacklist-based filter.

Parameters:

- **[Input]** *TextLength:* Length of the text in which the profanities are detected, measured in number of characters.
- **[Output]** *ProfanityCount:* Number of detected profanities alongside their location in the text.

Conversion Converts an MP3 file to a WAV file. This conversion tends to increase the file size, but many applications require raw WAV files as input. The conversion is performed using the Python library `pydub` (v0.23.1), a wrapper for the `ffmpeg` library, which is available in all Google Cloud function instances.

Parameters:

- **[Input]** *FileSize:* Size of the MP3 file that is converted.
- **[Output]** *FileSize:* Size of the resulting WAV file in bytes.

Censor Censors segments of a WAV file, based on a list of time segments that should be censored. For the censoring, all samples within the segments that are censored are muted using the `pydub` (v0.23.1) library.

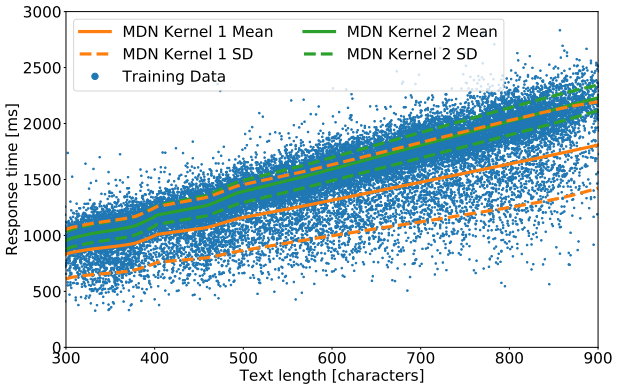


Figure 6: MDN model for the response time of the text2speech function.

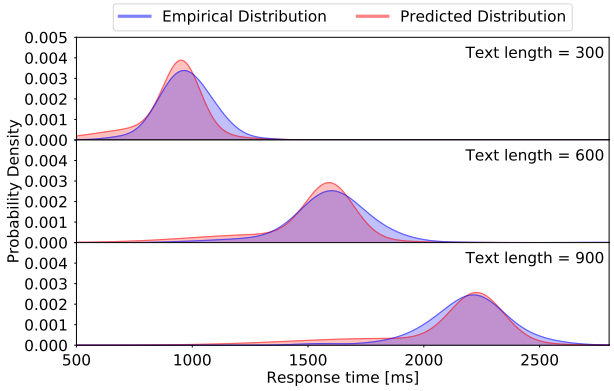


Figure 7: Comparison of measured response time distribution in the validation data set and predicted response time distribution for three different text lengths.

Parameters:

- **[Input]** *FileSize:* Size of the file that is censored in bytes.
- **[Input]** *ProfanityCount:* Number of detected profanities.
- **[Output]** *FileSize:* Size of the censored audio file in bytes.

Compression Compresses a WAV audio file by reducing the sampling rate and sample width using the `pydub` (v0.23.1) library. On an initial set of audio files, the compression achieves compression rates of about 60-90%.

Parameters:

- **[Input]** *FileSize:* Size of the audio file prior to compression, measured in bytes.
- **[Output]** *FileSize:* Size of the audio file after compression, measured in bytes.

We deploy each function to Google Cloud Functions with 512 MB memory, the Python 3.7 runtime and a timeout of 60 seconds. The code for the functions is available online[1]. In the following, we first investigate the capability of the proposed mixture density networks to accurately predict the distribution of a functions response time and its output parameters. Next, we apply our cost-prediction algorithm to two distinct workflows composed of these functions and compare the cost predictions to the actual observed costs.

3.1 Response time and output parameter distribution predictions

As described in Section 2.2, the mixture density networks can be trained on monitoring data collected during function operation or using micro-benchmarks. In this case study, we use micro-benchmarks to create the data set for the training of the mixture density networks as these functions are currently not deployed in production. The workload for each function consists of 50 requests/second with varying input parameters. The monitoring data for the first three minutes of a measurement is discarded as a warm-up phase. The next ten minutes are used as training data for the mixture density network models. For the evaluation in this paper, we additionally collect the monitoring data of the following 50 minutes as our validation data set. This larger validation data set is only required for the evaluation presented in this paper and is not necessary to apply our approach in practice. For the experiments presented in this paper, we parameterize the network as following. We use the Adam optimizer [30] with a learning rate of 0.001. We train for 500 epochs with a batch size of 32 and to prevent overfitting an early stopping criterion terminates the training process if the negative log-likelihood does not decrease by more than 0.001 for 10 epochs [42]. Additionally, we apply model check-pointing to save the best model achieved during training as the model accuracy decreased at times during the training process. In order to determine the appropriate number of kernels for the gaussian mixture model, we apply basic hyper-parameter optimization based on the observed negative log-likelihood during model training to select between 1, 2, 3, 4 or 5 kernels. The collected monitoring data, the implementation of the proposed approach and the evaluation scripts are available online as a CodeOcean capsule to enable 1-click reproduction of our results[2].

As an example for the resulting models, Figure 6 shows how the mixture density networks fit the training data for the response time of the Text2Speech function. The input parameter TextLength varies from roughly 300 to 900 characters and the resulting response time ranges from roughly 400 ms up to 3000 ms. There is a clear correlation between the length of the transcribed text and the response time of the Text2Speech function. However, for each input text length, a broad range of response times is observed. The mixture density network describes this distribution using two normal distributed kernels. The green normal distribution (MDN Kernel 2) is used to fit the bulk of occurring response times and the orange normal distribution (MDN Kernel 1) is used to describe the scattered lower response times. It is important to note, that these two distributions are not weighted equally, instead, the green kernel has a larger weight than the orange kernel.

Figure 7 shows how the predicted distributions (in red) compare to the observed empirical distributions from the validation data set for text lengths of 300, 600 and 900. To derive the empirical distributions we apply a gaussian kernel density estimation [12] with a bandwidth

[1] https://github.com/SimonEismann/FunctionsAndWorkflows

[2] https://doi.org/10.24433/CO.6374129.v2

Function	Parameter	1 kernel	2 kernels	3 kernels	4 kernels	5 kernels
Text2Speech	Response time	5.3%	4.2%	4.1%	6.4%	**4.5%**
Text2Speech	FileSize	0.6%	0.3%	1.1%	**0.4%**	0.6%
Conversion	Response time	13.2%	38.3%	3.4%	3.3%	**3.3%**
Conversion	FileSize	0.9%	**1.2%**	7.8%	9.0%	16.4%
Compression	Response time	13.1%	4.3%	5.2%	4.4%	**3.6%**
Compression	FileSize	0.2%	1.7%	0.4%	**0.2%**	3.5%
ProfanityDet	Response time	38.7%	32.9%	12.8%	9.4%	**4.6%**
ProfanityDet	ProfanityCount	14.5%	69.0%	12.8%	**12.3%**	14.0%
Censor	Response time	9.5%	10.1%	8.5%	**8.2%**	9.1%
Censor	FileSize	**1.0%**	0.6%	0.7%	1.5%	7.9%

Table 1: Relative Wasserstein distance [%] between validation dataset and predictions of MDNs with 1-5 kernels. Kernel count selected by hyperparameter optimization highlighted in bold.

of 0.4. The predicted distributions capture both the mean of the empirical distribution and the shape of the distribution well. The shape of the predicted distributions is slightly left heavy compared to a normal distribution due to the addition of the second kernel.

Next, we investigate the impact of the number of kernels used in the mixture density network. Table 1 shows the prediction error of mixture density networks with one to five kernels for the response time and output parameter distributions of the five functions. As a measure for the similarity of two distributions, we use the Wasserstein metric [5], which is defined for two distributions u and v as:

$$l(u, v) = \int_{-\infty}^{\infty} |U - V| \tag{3}$$

with U and V as the cumulative distribution function of u and v, respectively. Generally speaking, the Wasserstein metric quantifies how far a sample from a set of samples drawn from u has to be moved on average in order to transform the set of samples drawn from u to a set of samples drawn from v. As the absolute values of the Wasserstein metric are difficult to interpret, we calculate the relative Wasserstein metric by dividing the absolute Wasserstein metric by the mean of the empirical distribution as proposed in [34]. This relative Wasserstein metric enables us to quantify the prediction accuracy for a single input value. As a mixture density network predicts a different distribution for each input value, we calculate the weighted average over all values of the input distribution with the number of empirical samples as a weight.

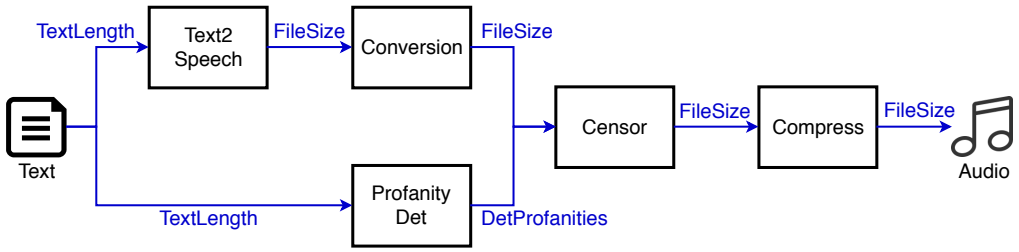
Table 1 shows the weighted average of the relative Wasserstein metric of mixture density networks with one to five kernels for the response time and output parameter distributions of the five functions and the kernel number selected by the hyper-parameter optimization. Generally, it seems that response time distributions are harder to predict than output parameter distributions. This is intuitive, as the output parameter of a function for a certain input is often constant, i.e., transcribing a text segment multiple times results in the same audio file each time, but the response time varies between executions. An outlier in this regard is the output parameter ProfanityCount of the function ProfanityDet, which has a higher error compared to the other parameters. This does not necessarily indicate a bad model fit as the target parameter ProfanityCount is an integer value of less than ten in most cases. The relative wasserstein metric assigns high percentage errors for even small deviations between integer distributions with a small range of values that also includes zero.

Regarding the kernel count, these results show that there is no kernel count that is ideal for every function. While three to five kernels seem to generally produce accurate performance predictions, the prediction error for the FileSize parameter of the conversion function with five kernels is 16.4%, while using two kernels results in a prediction error of 1.2%. This shows that selecting an individual number of kernels for each function is necessary. The hyper-parameter optimization based on the observed negative log-likelihood during model training reliably selects a kernel count that provides accurate predictions. In four out of ten scenarios, the hyper-parameter optimization does not select the ideal number of kernels, but the prediction accuracy of the selected kernels is always within one percentage point of the ideal kernel count.

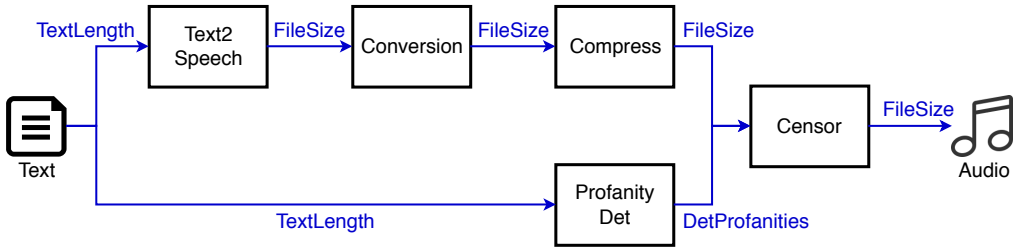
Across all functions, response time and parameter distributions, the mixture density network models selected by the hyper-parameter optimization achieve a prediction error of 3.9% and therefore a **prediction accuracy of 96.1%**. This shows that mixture density networks are capable of accurately predicting the distribution of the response time and the output parameters of a serverless function (**RQ1**).

3.2 Workflow cost predictions

For the evaluation of our workflow cost prediction algorithm, we consider the following scenario: A workflow designer is looking to build a workflow that turns short text segments into speech and censors any profanities within the text segment. For this task, he comes up with the two different workflows shown in Figure 8. In both workflows, the input text is first passed to the Text2Speech function and then converted to a WAV file using the Conversion function. In parallel, the text is also passed to the ProfanityDet function in order to identify any profanities within the text. In WorkflowA , any identified profanities are censored first using the Censor function and afterwards the audio file is compressed. In WorkflowB, the audio file is compressed prior to the censoring. While it is a reasonable



(a) WorkflowA.



(b) WorkflowB.

Figure 8: The two alternatives for the transcription and censoring workflow. The blue arrows indicate parameters passed to a function from a previous function.

Workflow	Metric	Invocations	CPU Time	Memory Time	**Total**
WorkflowA	Measured cost [cent]	$2.00 * 10^{-6}$	$8.60 * 10^{-5}$	$1.40 * 10^{-5}$	$\mathbf{1.02 * 10^{-4}}$
WorkflowA	Predicted cost [cent]	$1.79 * 10^{-6}$	$9.08 * 10^{-5}$	$1.35 * 10^{-5}$	$\mathbf{1.06 * 10^{-4}}$
WorkflowA	Relative prediction error	10.3%	5.5%	3.4%	**4.0%**
WorkflowB	Measured cost [cent]	$2.00 * 10^{-6}$	$3.80 * 10^{-5}$	$6.00 * 10^{-6}$	$\mathbf{4.60 * 10^{-5}}$
WorkflowB	Predicted cost [cent]	$1.79 * 10^{-6}$	$3.70 * 10^{-5}$	$5.52 * 10^{-6}$	$\mathbf{4.43 * 10^{-5}}$
WorkflowB	Relative prediction error	10.3%	2.6%	8.0%	**3.6%**

Table 2: Comparison between measured and predicted cost for a single workflow execution for both workflows in EUR.

assumption that WorkflowB might be cheaper, manually quantifying the cost difference is currently challenging for a workflow designer.

We apply the algorithm proposed in Section 2.3 in combination with the mixture density network models with two kernels from Section 3.1. To measure the actual execution cost of both workflow alternatives, we implement both workflows using Google Cloud Composer (a managed Apache Airflow service). The implementation of the workflows is also available online[3]. At the time of writing, the billing reporting for Google Cloud Functions is quite coarse-grained. An example of the most detailed reporting currently possible: On June 7th, 2019 you paid 43.7$ for 4,370,000 GHz-seconds CPU time of Cloud Functions, 30.5$ for 12,200,000 GB-seconds memory time of Cloud Functions and 5.5$ for 13,750,000 invocations of Cloud Functions. There is currently no option to report costs for time frames smaller than full days and no capability to report costs for a specific function or function execution. Additionally, no costs are reported until the free tier of 2 million invocations, 400,000 GB-seconds memory time and 200,000 GHz-seconds CPU time are used up.

Based on these limitations, we use the following approach to experimentally evaluate the costs of both workflows. First, we purposefully use up the capacity of the free tier by executing arbitrary functions. Next, we reserve a day for each experiment where no other functions are executed. During this day, we execute the first workflow 5,000 times with text segments with a normally distributed length ($\mu = 500$, $\sigma = 50$). At the start of the next day we take the aggregated costs for the day and divide them by 5,000 in order to get the average cost per workflow execution. We repeat the same process for the second workflow.

Table 2 shows the measured costs per workflow execution, the predicted costs using our approach and the resulting relative prediction error. At first glance, the measured prices seem unrealistically low. However, this is mostly due to the unfamiliar pricing scheme of cost per execution. If we were to assume that a n1-standard-2 VM (2 vCPU, 7.5GB memory) from Google Cloud (currently priced at

[3] https://github.com/SimonEismann/FunctionsAndWorkflows

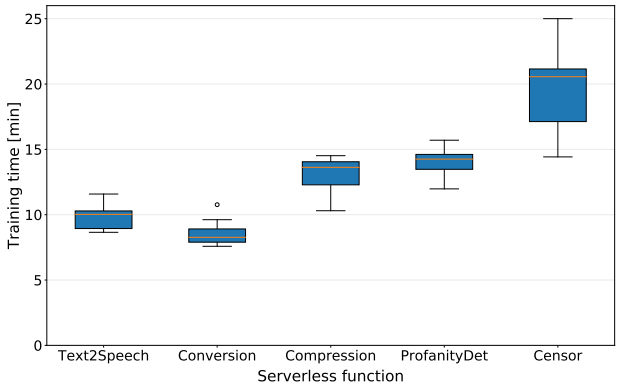


Figure 9: Training time for both models of each serverless functions with hyper-parameter optimization.

$0.0950 per hour) can handle 5 requests per second, this would result in a cost of $5.3 \cdot 10^{-6}$ per request. Consequently, costs of $1.02 \cdot 10^{-4}$ and $4.60 \cdot 10^{-5}$ to execute a workflow consisting of five functions is cheap, but within reason.

Existing approaches that rely on cost estimations for serverless functions and workflow simply round the mean observed function response time up to the nearest 100ms [13, 18]. Applying this cost estimation approach results in a cost estimate of $4.06 \cdot 10^{-5}$ for both workflows as it assumes that the function execution cost is independent of its context. However, a single execution of WorkflowA costs $1.02 \cdot 10^{-4}$ ct, whereas an execution of WorkflowB costs only about half as much even though both workflows provide the same functionality. This results in a relative prediction error of 60.2% for WorkflowA and 11.8% for WorkflowB using the naive cost estimate.

Our approach on the other hand accurately predicts this cost difference between the two workflows. It predicts the charged costs for invocations, CPU time and memory time with a prediction error ranging from 2.6% to 10.3%. An interesting observation is that the measured and predicted cost for the number of invocations differs. As in the workflows from our case study, the number of function invocations is static, our approach correctly predicts that 25,000 functions (5 executions per workflow * 5,000 workflow executions) will be executed. The billed costs on Google Cloud are rounded up to full cents, which causes this observed difference between measured and predicted costs for function invocations. We are looking to repeat the measurements with a larger number of workflow executions to significantly decrease the measurement uncertainty caused by the rounding of billed costs.

Overall, the costs for the execution of WorkflowA and WorkflowB are predicted with an error of 4.0% and 3.6% respectively, resulting in a average **cost prediction accuracy of 96.2%**. This shows that the proposed approach can accurately predict the expected costs of previously unobserved workflows (**RQ 2**).

3.3 Overhead analysis

The proposed approach enables accurate cost predictions for serverless workflows. In order to ensure that the proposed approach is applicable in practice, we investigate the time required to train the machine learning models for each function and the time required for the Monte-Carlo simulation.

The following experiments were conducted using a Intel® Core™ i5-4690K CPU with 3.50 Ghz. We measure the time required to train the MDN model for the response time distribution and the MDN model for the output parameter for each function from our case study with the hyper-parameter optimization to determine the appropriate number of kernels, which requires training MDN models. Figure 9 shows the result of repeating this measurement ten times as a boxplot. The training process for each function which includes training ten mixture density networks takes between 10 and 15 minutes, with the censor function as a small outlier with a median training time of 20 minutes. This difference can be attributed to an increased model complexity due to the additional input parameter. In general, we consider these training times acceptable as the training is performed offline and can be easily parallelized.

Additionally, we measure the time required to derive the cost predictions for a workflow using the Monte-Carlo simulation. Predicting the costs of WorkflowA requires 16.34±0.30 (N=10) seconds, whereas the predictions for WorkflowB require 14.20±0.03 (N=10) seconds. A user looking to compare these two workflow alternatives would need to wait about 30 seconds. Therefore, we consider the time requirements of using our approach in production feasible (**RQ3**).

4 LIMITATIONS

While our approach provides accurate cost predictions for serverless workflows in our case study, there still are limitations and threats to the validity to be discussed.

First, serverless functions can be provisioned with different memory limits, which indirectly also changes the processing power allocated to each function instance. Our approach currently does not take this into consideration and assumes that if a function is used in a workflow, its memory limit is not changed. While we consider this assumption reasonable, our approach could be combined with techniques that use transfer learning to determine the impact of configuration parameters on performance [27].

Besides costs for CPU time, memory time and a flat execution cost, cloud providers usually also charge for network egress, i.e., the amount of data leaving their data center or a regional zone. Our approach currently does not consider this type of costs as the specification of the workflow model does not contain any information about when data leaves a regional zone or the data center of the cloud provider. However, our approach is already capable of estimating the size of the output data of a serverless function and if the workflow model is extended accordingly, it should also be possible to predict the egress costs. However, this still needs to be validated in a further case study.

The approach proposed in this paper considers functions as black-boxes which can only be monitored at the interface level. Therefore, it does not explicitly model potential external calls within the serverless functions. For external calls to other serverless solutions such as serverless object storage (e.g., S3 Buckets or Google Cloud Storage), serverless databases (e.g. AWS Aurora or Google Cloud Datastore), serverless event management (e.g., AWS SNS or Google Cloud Pub/Sub) or serverless in-memory data storage (e.g.,

AWS Elasticache) should not impact the response time prediction accuracy, as the load-independent response time of these calls is correctly modeled within the mixture density network describing the response time of the function issuing the external call. External calls to non-serverless services can negatively impact the response time prediction accuracy, as the load-dependent behavior of these external calls is not captured by our approach. In practice, this should be neglectable as synchronous external calls in serverless functions are a major anti-pattern as they cause double-billing [8].

Lastly, our approach currently does not consider the costs for the Google Cloud Composer cluster used to execute the workflow. Google Cloud Composer charges for a set of VMs, whereas AWS Step Functions and Azure Logic Apps charge per connector within the workflow. For this pricing model, the static costs could be derived from our workflow model similar to how the static costs per execution are calculated. The Google Cloud Composer costs cannot be directly translated to costs per execution. If we can reliably estimate the number of workflows a cluster executes in a given time interval, it would be possible to translate the static hourly costs for the Google Cloud Composer cluster to a cost per workflow execution.

5 RELATED WORK

The existing publications related to this paper can be divided into three groups: discussions about the costs of serverless computing, cost prediction of serverless computing solutions and performance distribution prediction approaches.

Discussions about the costs of serverless computing. In the work of Adzic and Chatley [2], two industrial case studies of companies migrating from traditional hosting options to serverless computing are presented. The companies reported cost savings of 66% and 95% respectively after switching to serverless computing. The authors also discuss the non-constant response times of serverless functions as a limitation of current serverless platforms.

Eivy et al. claim that while the costs of serverless computing seem simple on the surface, they are surprisingly complicated in practice [17]. They discuss the issue of rounding up the function execution times to 100 ms and that response time estimates require deploying and testing the function. They also compare a serverless solution to traditional hosting in a case study with a large scale API, where the costs for the serverless solution are almost trice the costs for the VM based solution.

Vazquez et al. conducted a study on the applicability of serverless computing for data-intensive applications [39]. They compare a solution based on AWS Lamba to using EC2 to process data collected by the MARS Express orbiter from the European Space Agency. In their case study, both solutions incur similar costs, but the serverless solution is roughly twice as fast.

Performance distribution prediction. Khoshkbarforoushha et al. apply mixture density networks to predict the distribution of CPU time and execution time of Hive queries [29]. In this study, the mixture density networks achieved similar accuracy to state-of-the-art approaches concerning single point estimates and additionally accurate descriptions of the expected metric distribution.

Samani et al. apply mixture density networks to predict the distribution of service metrics based on infrastructure measurements. They report that while the predictions were surprisingly accurate, it also took considerable time to identify effective model parameterizations [36].

Cost prediction of serverless computing solutions. In the work of Boza et al. [13], an approach using model-based simulations to compare the costs of reserved VMs, on-demand VMs and serverless functions is introduced. The authors propose to model serverless functions as $M(t)/M/\infty$ queues, which assumes constant function response times and does not consider the impact of input parameters. The authors further conducted a survey with 96 participants, which revealed that many companies rely on reserved VMs to simplify financial planning.

Another approach to optimize the costs of serverless workflows by deciding whether to fuse multiple functions into a larger function and which memory limit should be allocated to a serverless function is proposed in the work by Elgmal [18]. This approach also relies on a constant value instead of a distribution for the response time of the serverless function and does not consider the impact of input parameters on the response time of a serverless function.

Gunasekaran et al. propose to use serverless functions in combination with VM-based hosting to enable SLO and cost-aware resource procurement [23]. To enable the cost-aware decision making between VM-based hosting and serverless functions, the authors also rely on cost-predictions for the serverless functions. This approach also relies on a constant value instead of a distribution for the response time of the serverless function and does not consider the impact of input parameters on the response time of a serverless function.

6 CONCLUSION

Serverless functions enable the execution of arbitrary functions, paying for usage rather than for reserved computing resources. To provide complex functionality, these serverless functions are often assembled into workflows. However, estimating the costs of these serverless workflow is challenging as the response time and therefore the costs of a serverless function depend on its input parameters, which are propagated from prior functions within the workflow. Existing approaches for the cost estimation of serverless functions and workflows do not take the influence of input parameters on the response time into account [13, 18]. In this paper, we propose methodolgy to predict the costs of serverless workflows. First, we apply mixture density networks to predict the distribution of a function's response time and its output parameters. The resulting models are then combined into a workflow model. Based on this workflow model, a Monte-Carlo simulation derives cost estimates for the workflow execution. The cost predictions provided by our approach enable workflow designers to evaluate and compare workflow alternatives, as well as optimize existing workflows. Our approach represents a first step towards fully-automated workflow optimization based on multi-objective optimization techniques. In a case study with two audio-processing workflows, our approach is able to predict the response time and output parameter distributions of five serverless functions with an accuracy of 96.1% and the costs of two workflow alternatives with an accuracy of 96.2%. As part of our future work, we will investigate approaches to predict the impact of different memory sizes on the performance of serverless functions.

ACKNOWLEDGEMENTS

This material is based upon work supported by Google Cloud. The authors would like to thank the anonymous reviewers for their valuable feedback and literature suggestions.

REFERENCES

[1] Vanessa Ackermann, Johannes Grohmann, Simon Eismann, and Samuel Kounev. 2018. Black-box Learning of Parametric Dependencies for Performance Models. In *Proceedings of 13th International Workshop on Models@run.time (MRT)*.

[2] Gojko Adzic and Robert Chatley. 2017. Serverless computing: economic and architectural impact. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 884–889.

[3] Aldeida Aleti, Stefan Bjornander, Lars Grunske, and Indika Meedeniya. 2009. ArcheOpterix: An Extendable Tool for Architecture Optimization of AADL Models. In *Proceedings of the 2009 ICSE Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES '09)*. IEEE Computer Society, 61–71.

[4] Amazon. 2018. Autodesk Goes Serverless in the AWS Cloud, Reduces Account-Creation Time by 99%. https://aws.amazon.com/solutions/case-studies/autodesk-serverless/. (2018). Accessed: 2019-05-28.

[5] Luigi Ambrosio, Nicola Gigli, and Giuseppe Savaré. 2008. *Gradient flows: in metric spaces and in the space of probability measures*. Springer Science & Business Media.

[6] Timon Back and Vasilios Andrikopoulos. 2018. Using a microbenchmark to compare function as a service solutions. In *European Conference on Service-Oriented and Cloud Computing*. Springer, 146–160.

[7] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, et al. 2017. Serverless computing: Current trends and open problems. In *Research Advances in Cloud Computing*. Springer, 1–20.

[8] Ioana Baldini, Perry Cheng, Stephen J. Fink, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Philippe Suter, and Olivier Tardieu. 2017. The Serverless Trilemma: Function Composition for Serverless Computing. In *Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! 2017)*. ACM, 89–103.

[9] Christopher M Bishop. 1994. *Mixture density networks*. Technical Report.

[10] Egor Bondarev, Peter de With, Michel Chaudron, and Johan Muskens. 2005. Modelling of input-parameter dependency for performance predictions of component-based embedded systems. In *31st EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, 36–43.

[11] Oliver Borchers. 2015. A Hitchhiker's Guide to Mixture Density Networks. https://towardsdatascience.com/a-hitchhikers-guide-to-mixture-density-networks-76b435826cca. (2015). Accessed: 2019-05-28.

[12] Zdravko I Botev, Joseph F Grotowski, Dirk P Kroese, et al. 2010. Kernel density estimation via diffusion. *The annals of Statistics* 38, 5 (2010), 2916–2957.

[13] Edwin F Boza, Cristina L Abad, Mónica Villavicencio, Stephany Quimba, and Juan Antonio Plaza. 2017. Reserved, on demand or serverless: Model-based simulations for cloud budget planning. In *2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM)*. IEEE, 1–6.

[14] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). (2015). arXiv:arXiv:1511.07289

[15] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. 2009. L 2 regularization for learning kernels. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 109–116.

[16] Simon Eismann, Jürgen Walter, Jóakim von Kistowski, and Samuel Kounev. 2018. Modeling of parametric dependencies for performance prediction of component-based software systems at run-time. In *2018 IEEE International Conference on Software Architecture (ICSA)*. IEEE, 135–13509.

[17] Adam Eivy. 2017. Be wary of the economics of" Serverless" Cloud Computing. *IEEE Cloud Computing* 4, 2 (2017), 6–12.

[18] Tarek Elgamal. 2018. Costless: Optimizing cost of serverless computing through function fusion and placement. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 300–312.

[19] Erwin Van Eyk, Lucian Toader, Sacheendra Talluri, Laurens Versluis, Alexandru Uta, and Alexandru Iosup. 2018. Serverless is More: From PaaS to Present Cloud Computing. *IEEE Internet Computing* 22, 5 (2018), 8–17.

[20] DN Geary. 1989. *Mixture Models: Inference and Applications to Clustering*. Vol. 152. Royal Statistical Society. 126–127 pages.

[21] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. PMLR, 315–323.

[22] Johannes Grohmann, Simon Eismann, Sven Elflein, Manar Mazkatli, Jóakim von Kistowski, and Samuel Kounev. 2019. Detecting Parametric Dependencies for Performance Models Using Feature Selection Techniques. In *Proceedings of the 27th IEEE International Symposium on the Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '19)*. IEEE, 309–322.

[23] Jashwant Raj Gunasekaran, Prashanth Thinakaran, Mahmut Taylan Kandemir, Bhuvan Urgaonkar, George Kesidis, and Chita Das. 2019. Spock: Exploiting serverless functions for slo and cost aware resource procurement in public cloud. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 199–208.

[24] Joseph M. Hellerstein, Jose Faleiro, Joseph E. Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu. 2018. Serverless Computing: One Step Forward, Two Steps Back. (2018). arXiv:arXiv:1812.03651

[25] Scott Hendrickson, Stephen Sturdevant, Tyler Harter, Venkateshwaran Venkataramani, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2016. Serverless Computation with openLambda. In *Proceedings of the 8th USENIX Conference on Hot Topics in Cloud Computing (HotCloud'16)*. USENIX Association, 33–39.

[26] IBM. 2017. Serverless Architectures in Banking: OpenWhisk on IBM Bluemix at Santander. https://developer.ibm.com/code/videos/tech-talk-replay-build-faster-banking-apps-ibm-cloud-functions/. (2017). Accessed: 2019-05-28.

[27] Pooyan Jamshidi, Norbert Siegmund, Miguel Velez, Christian Kästner, Akshay Patel, and Yuvraj Agarwal. 2017. Transfer learning for performance modeling of configurable systems: An exploratory analysis. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 497–508.

[28] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar, Joseph E. Gonzalez, Raluca Ada Popa, Ion Stoica, and David A. Patterson. 2019. Cloud Programming Simplified: A Berkeley View on Serverless Computing. (2019). arXiv:arXiv:1902.03383

[29] Alireza Khoshkbarforoushha and Rajiv Ranjan. 2016. Resource and performance distribution prediction for large scale analytics queries. In *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*. ACM, 49–54.

[30] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. (2014). arXiv:arXiv:1412.6980

[31] Klaus Krogmann, Michael Kuperberg, and Ralf Reussner. 2010. Using genetic search for reverse engineering of parametric behavior models for performance prediction. *IEEE Transactions on Software Engineering* 36, 6 (2010), 865–877.

[32] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436.

[33] Theo Lynn, Pierangelo Rosati, Arnaud Lejeune, and Vincent Emeakaroha. 2017. A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 162–169.

[34] Szymon Majewski, Michal Aleksander Ciach, Michal Startek, Wanda Niemyska, Blazej Miasojedow, and Anna Gambin. 2018. The Wasserstein Distance as a Dissimilarity Measure for Mass Spectra with Application to Spectral Deconvolution. In *18th International Workshop on Algorithms in Bioinformatics (WABI 2018) (Leibniz International Proceedings in Informatics (LIPIcs))*, Laxmi Parida and Esko Ukkonen (Eds.), Vol. 113. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 25:1–25:21.

[35] Anne Martens, Heiko Koziolek, Steffen Becker, and Ralf Reussner. 2010. Automatically Improve Software Architecture Models for Performance, Reliability, and Cost Using Evolutionary Algorithms. In *Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering (WOSP/SIPEW '10)*. ACM, 105–116.

[36] Forough Shahab Samani and Rolf Stadler. 2018. Predicting distributions of service metrics using neural networks. In *2018 14th International Conference on Network and Service Management (CNSM)*. IEEE, 45–53.

[37] Erwin van Eyk, Johannes Grohmann, Simon Eismann, André Bauer, Laurens Versluis, Lucian Toader, Norbert Schmitt, Nikolas Herbst, Cristina L. Abad, and Alexandru Iosup. [n. d.]. The SPEC-RG Reference Architecture for FaaS: From Microservices and Containers to Serverless Platforms. *IEEE Internet Computing* ([n. d.]). https://doi.org/10.1109/MIC.2019.2952061

[38] Erwin Van Eyk, Alexandru Iosup, Cristina L Abad, Johannes Grohmann, and Simon Eismann. 2018. A SPEC RG cloud group's vision on the performance challenges of FaaS cloud architectures. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*. ACM, 21–24.

[39] Jose Luis Vazquez-Poletti, Ignacio Martín Llorente, Konrad Hinsen, and Matthew Turk. 2018. Serverless computing: from planet mars to the cloud. *Computing in Science & Engineering* 20, 6 (2018), 73–79.

[40] Laurens Versluis, Erwin Van Eyk, and Alexandru Iosup. 2018. An Analysis of Workflow Formalisms for Workflows with Complex Non-Functional Requirements. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*. ACM, 107–112.

[41] David Vose. 2008. *Risk analysis: a quantitative guide*. John Wiley & Sons.

[42] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. 2007. On early stopping in gradient descent learning. *Constructive Approximation* 26, 2 (2007), 289–315.

[43] Qingchen Zhang, Laurence T Yang, Zhikui Chen, and Peng Li. 2018. A survey on deep learning for big data. *Information Fusion* 42 (2018), 146–157.