MetaFaaS: learning-to-learn on serverless

Conference Paper · June 2022

DOI: 10.1145/3530050.3532926

6 authors, including:

Varad Pimpalkhute
University of Massachusetts Amherst
9 PUBLICATIONS **116** CITATIONS
SEE PROFILE

Shruti Kunde
Tata Consultancy Services Limited
24 PUBLICATIONS **93** CITATIONS
SEE PROFILE

Rekha Singhal
Tata Consultancy Services Limited
97 PUBLICATIONS **299** CITATIONS
SEE PROFILE

Dheeraj Chahal
Clemson University
35 PUBLICATIONS **121** CITATIONS
SEE PROFILE

MetaFaaS: Learning-to-learn on Serverless

Varad Pimpalkhute, Shruti Kunde, Rekha Singhal, Surya Palepu, Dheeraj Chahal, Amey Pandit
{varad.p,shruti.kunde,rekha.singhal,surya.palepu,d.chahal,amey.panditl}@tcs.com

ABSTRACT

Meta-learning is a technique to transfer learning from a pre-built model on known tasks to build a model for unknown tasks. Graident-based meta-learning algorithms are one such family that use the technique of gradient descent for model updates. These meta-learning architectures are hierarchical in nature and hence incur large training times, which are prohibitive for industries relying on models trained using the most recent data to make relevant predictions. To address these issues, we propose MetaFaaS, a function-as-a-service (FaaS) paradigm on public cloud to build a scalable and cost-performance optimal deployment framework for gradient-based meta-learning architectures. We propose an analytical model to predict the cost and training time on cloud for a given workload. We validate our approach on multiple meta-learning architectures, (MAML, ANIL, ALFA) and attain a speed-up of over 5× in training time on FaaS. We also propose eALFA, a compute-efficient meta-learning architecture, which achieves a speed-up of > 9× as compared to ALFA. We present our results with four quasi-benchmark datasets in meta-learning, namely, Omniglot, Mini-Imagenet (Imagenet), FC100 (CIFAR), and CUBirds200.

CCS CONCEPTS

• **Computing methodologies** → **Distributed algorithms**.

KEYWORDS

meta-learning, serverless, distributed training, few-shot-learning

ACM Reference Format:
Varad Pimpalkhute, Shruti Kunde, Rekha Singhal, Surya Palepu, Dheeraj Chahal, Amey Pandit. 2022. MetaFaaS: Learning-to-learn on Serverless. In *Big Data in Emergent Distributed Environments (BiDEDE'22), June 12, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3530050.3532926

1 INTRODUCTION AND MOTIVATION

One of the biggest challenges with machine learning (ML) or deep learning (DL) models is the assumption that, once they are deployed, they will continue to perform well forever. It is inevitable that data changes over time, subsequently affecting the model performance adversely. Domains such as automatic speech recognition (ASR) [9, 32] and medical imaging [14, 24], use large and complex pre-trained models for predictions. The models need to be continually re-trained when few, albeit unseen audio sample files or new medical images (CT Scans, X-Rays, Skin lesions, etc) are encountered. Another domain, namely FinTech [7, 12, 13], frequently uses stress-testing for scenarios involving risk-calculation, compliance checks and fraud-detection. Such scenarios require frequent (re)training of models, as data comes in at intervals. A meta-learning approach, also known as learn-to-learn or few-shot paradigm [6], is ideally suited for training models to generalize on new unseen tasks using just a *few examples*. In contrast, deep learning works best when a high volume of good quality data is available and performance improves as the data grows. However, a challenge with meta-learning architectures is the large training times incurred, especially by gradient based algorithms (a widely used school of thought in meta-learning) owing to their hierarchical nature. In [16] the authors have explored the idea of distributed training for accelerating meta-learning architectures on a bare-metal setup. This results in a limited scalability of the application and incurs of a fixed setup cost.

With the advent of cloud providers and their offerings, virtual machines (VMs) can be provisioned to facilitate scalability and accelerate the process of distributed training.

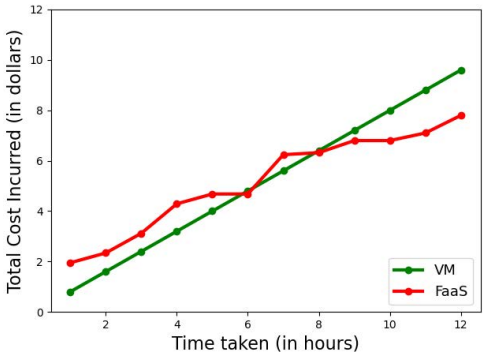


Figure 1: Cost incurred on VMs vs Serverless

However in the Fintech stress-testing scenario outlined above, data arrives at intervals and the model needs to be retrained each time. A virtual machine will need to be continuously provisioned. When there is no incoming data, the VM will stay idle or remain under-utilized, while the cost incurred keeps increasing. Function-as-a-Service (FaaS) [18] or serverless architecture is a cost-effective solution, that enables better scalability by adjusting and tuning the number of servers as the business grows. Figure 1 illustrates the utility of FaaS architectures in such scenarios, that enable cost optimization.

Data comes in at irregular intervals (Figure 1), and over time (refer the x-axis), the cost incurred when using VMs drastically increases as the VMs need to stay connected, while a serverless setup is cost-effective as one pays only for whatever is used, i.e.,

BiDEDE '22, June 12–17, 2022, Philadelphia, PA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/3530050.3532926

unlike traditional cloud providers, FaaS providers do not charge for idle computation time [2]. Another issue is that gradient-based meta-learning algorithms are compute-intensive, thus increasing the overall training time [25, 28, 36], both on bare-metal and a serverless setup. The challenge lies in optimizing meta-learning architectures for compute-efficient performance, to improve the training time.

To address the challenges discussed above, we propose MetaFaaS, a meta-learning based scalable architecture using serverless distributed setup. We leverage the hierarchical nature of gradient-based architectures to facilitate distributed training on a serverless setup. We propose our compute-efficient architecture, eALFA, for meta-learning, with a significantly reduced training time. While FaaS enables unlimited scalability, we observe that there is an optimal number of FaaS instances that can reduce the training time. We propose a generic analytical model for gradient based meta-learning architectures for predicting the number of FaaS instances (and hence the cost incurred) necessary for minimizing the overall training time and validate it using quasi-benchmark datasets in the field of meta-learning. To summarize, our key contributions in this paper are as follows

(1) MetaFaaS : Cost-performance optimal deployment of gradient based meta-learning architectures using FaaS.
(2) eALFA : An efficient version of ALFA, that provides improved accuracy and optimal training times.
(3) Analytical model for performance evaluation of meta-learning architectures on FaaS.

The rest of the paper is structured as follows. We discuss the related work in Section 2. We present the MetaFaaS architecture in Section 3. Section 4 details the setup for the experiments and the results are presented in Section 5. We present the analytical model in Section 6. Lastly, we conclude with directions for future work in Section 7.

2 RELATED WORK

Meta-learning, also known as learning-to-learn has three schools of thought: model-based [20, 23, 31], metric-based [15, 27, 34, 35], and gradient-based [4, 8, 37]. Among them, gradient-based meta-learning algorithms are receiving increasingly more attention due to their potential to generalize across different tasks. A popular gradient-based algorithm namely, MAML [8] and its variants [3, 28] focuses on finding a good initialization of parameters, which is necessary for a model to converge quickly on unseen tasks at test time. However, the hierarchical nature of gradient based algorithms, makes them compute-intensive increasing the training time. Multiple efforts [5, 16] and [25, 28, 36] have proposed methodologies from systems and algorithmic perspectives respectively to accelerate or decrease the computation required for the meta-learning training process. [16] distributed meta-algorithms [8, 25, 28] on bare-metal using Ray [22] and Horovod [26] to achieve a speedup of 3x benchmarked against serial setup for the same set of algorithms. However, [16] is an on-premise approach, thus, it is limited in terms of scalability and is usually not very cost-effective.

With the advent of multiple cloud providers [2, 10, 19], ML/DL models can be trained in a highly cost-effective and scalable manner by provisioning VMs or cloud clusters in a distributed training setup. Recent works [33],[38] propose communication and/or resource efficient distributed training for ML models. A cost-optimal approach for easy deployment and scalability is FaaS. A few efforts [30] in the literature focus on using the FaaS architecture to provide a cost-efficient, resource provisioning framework, enabling predictable performance for ML/DL workloads. A recent work [11], builds a framework for implementing federated learning using FaaS, which is slower albeit cheaper and more resource efficient in the long run. In spite of meta-learning architectures being compute-intensive and having relatively large training times, we have not come across any work in the literature, that employs FaaS to accelerate the training or re-training process of meta-learning architectures.

3 METAFAAS ARCHITECTURE

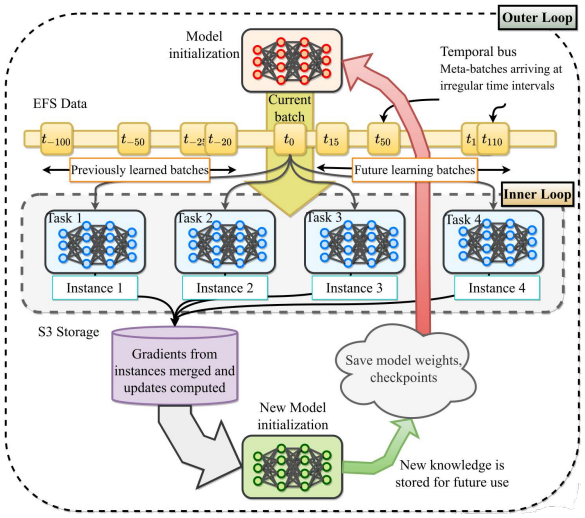


Figure 2: The MetaFaaS Architecture

We propose MetaFaaS (Figure 2), a generic serverless architecture for accelerating meta-learning algorithms. [1] Tasks are distributed across serverless instances and a copy of the model is trained at each instance. The storage, AWS S3 (in this case), will consolidate gradients from all workers, merge gradients and return the updated model parameters. Data at each worker can be read from any file system on the cloud. (eg: Elastic File System provided by AWS.) The data may be temporal in nature and arrive at irregular intervals as shown in the figure. We now outline how the meta-learning algorithm is distributed using MetaFaaS to optimize the training time and achieve scalability.

Algorithm 1 outlines a generic view of a gradient based meta-learning algorithm which have a hierarchical structure. They usually comprise of two loops (1) Inner loop (2) Outer loop. A *meta-batch* represents the number of tasks that will be processed in each iteration of the inner loop. A task [8] represents a distribution over input data samples, their corresponding labels and the loss function. Inside the inner loop, weight updates are collected from each task, and each set of weight updates will update parameters

[1] A serverless architecture is an event-driven computing service, where the developer can run a service in the backend without provisioning or managing servers.

Algorithm 1 Gradient based meta-learning

for $outerloop = 1, 2, \ldots$ **do**
  for $innerloop = 1, 2, \ldots, metabatchsize$ **do**
    Base model adapts to a new task.
    Task specific parameters are updated.
    Compute adapted parameters with gradient descent.
  end for
  Update base model/neural network parameters with gradient descent.
end for

of the model. The outer loop calculates the loss for each model (from the inner loop), determines gradients and updates the model parameters. MetaFaaS architecture (Figure 2) depicts multiple workers, i.e., serverless instances. For every inner loop iteration, a task is loaded on the instance and the model copy at that instance is trained. Thus, each worker will train a copy of the model, using data (tasks) present at the worker. *This differs from distributed training; where a model is trained on each instance with the batch of data that is loaded on the instance at the beginning of all epochs. For each epoch, batches are sampled from the subset of data present at each instance.* In contrast, MetaFaaS loads tasks at each instance from the meta-batch. A new meta-batch of tasks is loaded at the end of an epoch. The tasks are distributed across workers. Serverless instances cannot communicate with each other, hence the gradients are consolidated using any storage (eg: S3 in this case) mechanism. Model parameters are updated (outer loop) and updated copies of the model parameters are sent to all workers. This signifies the end of an epoch (also known as the outer loop). In this manner, the model is trained to convergence.

Next, we discuss general terminology in the field of meta-learning and then outline 3 popular meta-learning architectures, namely, MAML, ANIL and ALFA, followed by our proposed compute-efficient architecture, eALFA.

3.1 Preliminaries

Meta-learning is a few-shot learning paradigm to efficiently learn on unseen tasks given very few samples during training. A well-known meta-learning architecture, **Model-agnostic Meta Learner (MAML)** [8] tries to find a good initialization point for the model parameters. MAML performs two optimizations via two loops: Outer Loop and Inner Loop. In Inner Loop, MAML performs task-specific updates. It performs gradient update steps using SGD for input training samples of each task.

Another architecture, **ANIL - Almost No Inner Loop** [28] is a simplified version of MAML, where the parameter updates of the inner loop are considered redundant. ANIL thus removes the inner loop updates for the network body and applies inner loop adaptation only to the head. This is because, the head is task-specific, and thus varies each inner loop iteration to align with different classes in each task.

Because of its similarity with MAML, ANIL can easily be adapted to the serverless architecture, similar to MAML. The tasks in the meta-batch are distributed across multiple workers. However since computation in the inner-loop is already optimized (only the head layer is updated), we do not expect to see a very high speed-up in the distributed serverless setup for ANIL.

ALFA [3] is a gradient-based meta-learning algorithm, that focuses on adaptive learning of hyperparameters for fast adaptation, i.e., inner-loop optimization. It achieves this by making the weight decay and learning rate hyper-parameters adaptive, to the current state of the base learner. ALFA is initialization-agnostic because the initial weights θ for f_θ do not need to be updated throughout the training process, i.e., ALFA can be trained to adapt from any given initialization (e.g., random initializations). Naturally, ALFA can be used with a technique such as MAML that searches for the best model initialization to get even better performance. A detailed study of all the three meta-learning algorithms MAML, ANIL, and ALFA is discussed in Appendix A sub-sections A.1, A.2 and A.3 respectively.

3.2 eALFA

We now propose a compute-efficient variant of ALFA called efficient ALFA (eALFA). The ALFA algorithm is known to have a huge computational overhead due to the inclusion of an additional neural network in the training loop (Refer Figure 3a). The challenge is to reduce the computational overhead while maintaining the advantage of rapid learning using ALFA. Our experiments in freezing layers of the neural network gave us insights into parameters that were updated in the inner-loop training. CCA Similarity [21][2] was used to validate the training.

Proposed Algorithm

We trained ALFA on FC100 dataset using a 4-layered CNN (CONV4) having a classifier layer as a head for 100 iterations. In each iteration, we computed the CCA Similarity of the model parameters before and after the inner loop update. Figure 3c shows the results of our CCA Similarity experiments on the CONV4 model. As seen the CCA score for all four layers is above 0.9, meaning that the weights are not updated significantly in the inner loop. However, the head layer shows a low CCA score. This is because, the classes change for every input task, thus the head layer has to adapt to the classes. Thus, during the inner loop, we can freeze all the layers except the head layer, and still achieve nearly the same accuracy on the input dataset as shown in Equation 1:

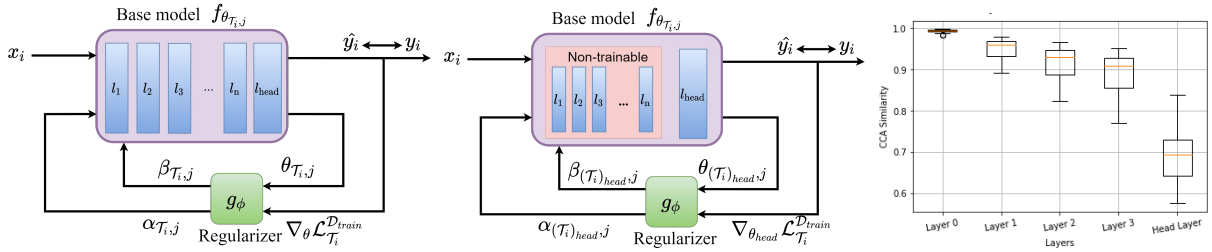
$$\begin{aligned} \theta^{l_k}_{\mathcal{T}_i} &= \beta_{l_k}\theta^{l_k}_{\mathcal{T}_i} - \alpha_{l_K}\nabla_\theta \mathcal{L}^{\mathcal{D}_{\text{train}}}_{\theta_{l_{\mathcal{T}_i}}}(f_\theta), \forall k = \{1, 2, \ldots N\} \\ \theta^{head}_{\mathcal{T}_i} &= \beta_{head}\theta^{head}_{\mathcal{T}_i} - \alpha_{head}\nabla_\theta \mathcal{L}^{\mathcal{D}_{\text{train}}}_{\theta_{l_{\mathcal{T}_i}}}(f_\theta) \end{aligned} \tag{1}$$

we would only update the head layer as in Equation 2

$$\begin{aligned} \theta^{l_k}_{\mathcal{T}_i} &= \theta^{l_k}_{\mathcal{T}_i}, \forall k = \{1, 2, \ldots N\} \\ \theta^{head}_{\mathcal{T}_i} &= \beta_{head}\theta^{head}_{\mathcal{T}_i} - \alpha_{head}\nabla_\theta \mathcal{L}^{\mathcal{D}_{\text{train}}}_{\theta_{l_{\mathcal{T}_i}}}(f_\theta) \end{aligned} \tag{2}$$

The eALFA algorithm is described in Algorithm 2.

[2]CCA Similarity is a technique that makes use of linear combinations of neurons present in activations of two given layers and maximizes the correlation between the two layers.



(a) ALFA. (b) eALFA. (c) CCA Similarity

Figure 3: (a,b) Architecture for ALFA, eAFlfa. (c) CCA Similarity plot

Algorithm 2 Efficient ALFA (eALFA)

Require: A Task Distribution $\mathcal{P}(\mathcal{T})$, learning rate γ

Randomly initialize θ and ϕ
while not DONE **do**
  Sample batches of tasks $\mathcal{T}_i \sim \mathcal{P}(\mathcal{T})$
  for tasks $\mathcal{T}_i$ **do**
    Initialize $\theta_{\mathcal{T}_i,0} = \theta$
    $\theta_1, \theta_2, ..., \theta_{\text{head}} = \theta_{\mathcal{T}_i,0}$
    Randomly sample two sets: $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}} \in \mathcal{T}_i$
    for adaptation steps $j := 0$ to AS $- 1$ **do**
      Compute loss $\mathcal{L}_{\mathcal{T}_i}^{\mathcal{D}_{\text{train}}}(f_{\theta_{\mathcal{T}_i,0}})$ w.r.t. $\mathcal{K} \in \mathcal{D}_{\text{train}}$
      Compute task-specific hyperparameters:
      $(\alpha_{\mathcal{T}_i,j}, \beta_{\mathcal{T}_i,j}) = g_\phi(\nabla_{\theta_{\text{head}}} \mathcal{L}_{\mathcal{T}_i}^{\mathcal{D}_{\text{train}}}(f_{\theta_{\mathcal{T}_i,j}}), \theta_{\text{head},j})$
      Perform gradient descent on the head layer:
      $\theta_{\text{head},j+1} = \beta_{\mathcal{T}_i,j}\theta_{\text{head},j+1} - \alpha_{\mathcal{T}_i,j}\mathcal{L}_{\mathcal{T}_i}^{\mathcal{D}_{\text{train}}}(f_{\theta_{\mathcal{T}_i,j}})$
    end for
    Compute loss $\mathcal{L}_{\mathcal{T}_i}^{\mathcal{D}_{\text{test}}}(f_{\theta'_{\mathcal{T}_i,j}})$ w.r.t. $\mathcal{K} \in \mathcal{D}_{\text{test}}$
    Update weights: $\theta'_{\mathcal{T}_i} = \theta_{\mathcal{T}_i,AS}$
  end for
  Perform gradient-descent on regularizer:
  $\phi := \phi - \gamma\nabla_\phi \sum_{\mathcal{T}_i} \mathcal{L}_{\mathcal{T}_i}^{\mathcal{D}_{\text{test}}}(f_{\theta'_{\mathcal{T}_i}})$
end while

4 EXPERIMENT SETUP

We benchmark our results for multiple quasi-benchmark datasets (Omniglot, Miniimagenet, FC100, CUBirds) in the field of meta-learning. We compare the performance of meta-learning architectures on bare-metal, with serverless. For the serverless architecture setup, we use AWS lambda instances. We study the performance by varying the number of instances (1,2,4,8,16) to increase parallelism during the training process. Our bare-metal experiments are conducted on a Linux CentOS7 server, with 256GB RAM and 56 core CPUs. The AWS lambda instance has 6 cores and a maximum memory of 10 GB can be allocated to the instance. Memory usage varies according to the input task and can be varied in the range of 128 MB to 10 GB to avoid additional costs. Each lambda instance can execute for 15 minutes once it is instantiated. We fix the metatbatch size to 16 (i.e., there are 16 tasks in each batch) and we conduct experiments with 1shot-5ways and 5shots-1way. The Mini-Imagenet dataset [29, 35] comprises 60K images of size 84x84. The dataset has 64 training classes, 16 validation classes and 20 testing classes, each having 600 samples. Omniglot [17] dataset is made up of 50 alphabets belonging to 1623 character classes, each containing 20 samples. The default setup of the learn2learn library[3] has 1100 classes in the train set, 100 classes for validation and 428 classes in the test set. We have used this default setup for our experiments. The CU-Birds [39] dataset contains 11,788 images of 200 bird species. The data is split into 200 classes that are divided into 100, 50 and 50 for meta-training, meta-validation and meta-testing, respectively. FC100 is a few-shot classification dataset built on CIFAR100 [27]. The dataset is split into 100 classes which are divided into 60 classes for meta-training, 20 classes for meta-validation, 20 classes for meta-testing, with each class containing 20 images.

5 RESULTS AND DISCUSSION

We conduct an extensive study with multiple meta-learning architectures (MAML, ANIL, ALFA and eALFA), across image datasets from varied domains. We benchmark the performance on bare-metal where the algorithms are trained in a serial setup, i.e., the model is trained sequentially on tasks in a meta-batch using two configurations (1) 5 ways, 1 shot (2) 5 ways, 5 shots, on increasing number of serverless lambda instances (1, 2, 4, 8).

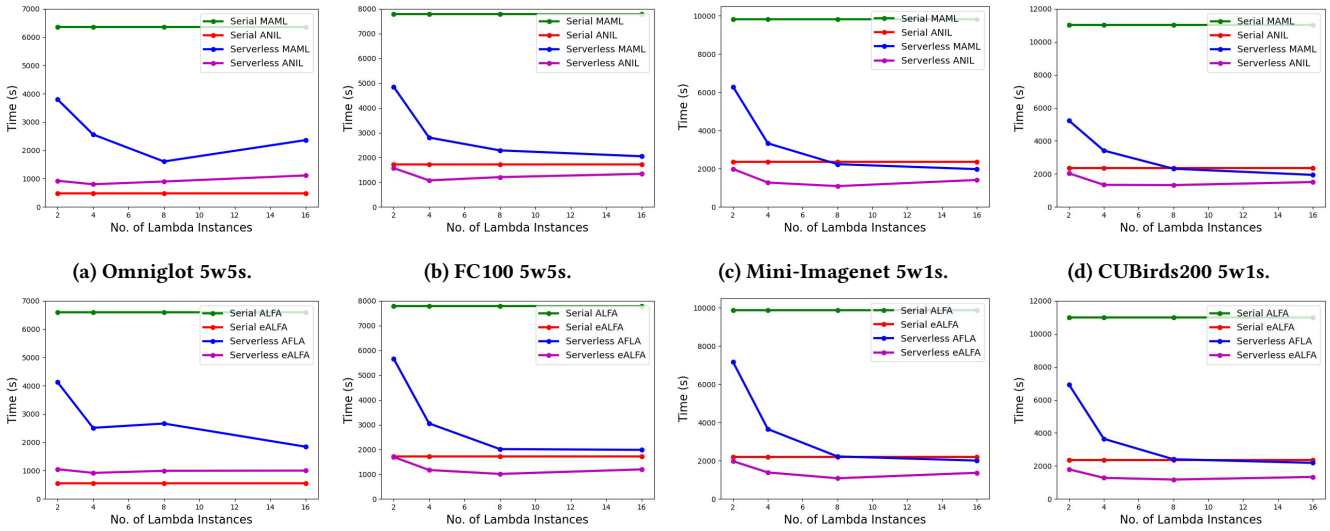
5.1 Performance acceleration on serverless

In this experiment, we compare the performance of MAML and ANIL using the serverless setup against a serial implementation on bare-metal. We plot the training time achieved for MAML running in a serial manner on a bare-metal setup and compare this with the serverless execution of the MAML and ANIL architectures on an increasing number of serverless instances on 4 datasets (Figure 4). As the number of instances increase, we observe that serverless MAML achieves an acceleration of more than 5X as compared to serial MAML. Serverless ANIL performs even better. This is because by default ANIL is computationally optimal as compared to MAML. However as the number of instances goes to 16, we observe an increase in the training time for small datasets, because the communication overhead for consolidating gradients at the end of every epoch increases. Thus, there is a trade-off between the speed-up in training time with scalability and communication overhead. Additional experiments are presented in Appendix C.1

[3] https://github.com/learnables/learn2learn/

Table 1: Accuracy comparision - Omniglot (OG) and FC100 (FC), 5shots/1shot on Serverless and Sr(Serial)

		MAML					ANIL					ALFA					eALFA				
		2w	4w	8w	16w	Sr	2w	4w	8w	16w	Sr	2w	4w	8w	16w	Sr	2w	4w	8w	16w	Sr
OG	1s	0.92	0.91	0.92	0.92	0.94	0.87	0.86	0.87	0.85	0.94	0.94	0.95	0.84	0.95	0.96	**0.96**	0.96	0.96	0.96	0.96
	5s	0.94	0.92	0.91	0.91	0.97	0.89	0.89	0.88	0.87	0.96	0.99	0.99	0.98	0.98	0.99	0.99	**0.99**	0.99	0.99	0.99
FC	1s	0.36	0.35	0.36	0.35	0.36	0.33	0.32	0.34	0.35	0.38	**0.39**	0.37	0.38	0.37	0.41	0.34	0.34	0.34	0.35	0.37
	5s	0.45	0.44	0.45	0.45	0.49	0.35	0.34	0.34	0.34	0.47	0.50	0.51	**0.51**	0.50	0.53	0.49	0.49	0.49	0.49	0.50



(a) Omniglot 5w5s. **(b) FC100 5w5s.** **(c) Mini-Imagenet 5w1s.** **(d) CUBirds200 5w1s.**

(e) Omniglot 5w5s. **(f) FC100 5w5s.** **(g) Mini-Imagenet 5w1s.** **(h) CuBirds200 5w1s.**

Figure 4: Comparison of performance of meta-learning algorithms on FaaS and Serial Setup.

We now benchmark the performance of our proposed approach eALFA and illustrate how eALFA is computationally optimal leading to a significant reduction in the training time. As observed from the graphs (Figure 4), ALFA shows an improvement in training time with an increase in serverless instances, while eALFA, outperforms ALFA on a serverless setup. As discussed in Section 3.2, eALFA updates only the head layer of the network and as a result reduces the compute overhead. This also leads to a further reduction in training time.

From Figure 4, we also infer that as the difficulty of input tasks increases, the speedup gained on a serverless platform also increases. We obtain a speedup of more than 5x for MAML and ALFA meta-learning algorithms for datasets such as Mini-Imagenet and CUBirds. We notice a communication overhead for ANIL and eALFA datasets for easier datasets such as Omniglot. However, in general, we achieve a speedup of 2x-4x for MAML and ALFA algorithms, and a speedup of 1.5x-2.5x for ANIL and eALFA algorithms.

Table 1 presents the comparison between the accuracy achieved on serial bare-metal setup versus the accuracy observed across different instances on serverless for Omniglot (OG) and FC100 (FC) datasets, for 1 shot (1s) and 5 shot (5s). We observe that the model performance is not affected significantly even when subject to multiple instances. Furthermore, we achieve a huge speedup on different meta-learning architectures trained across various datasets.

6 ANALYTICAL MODEL

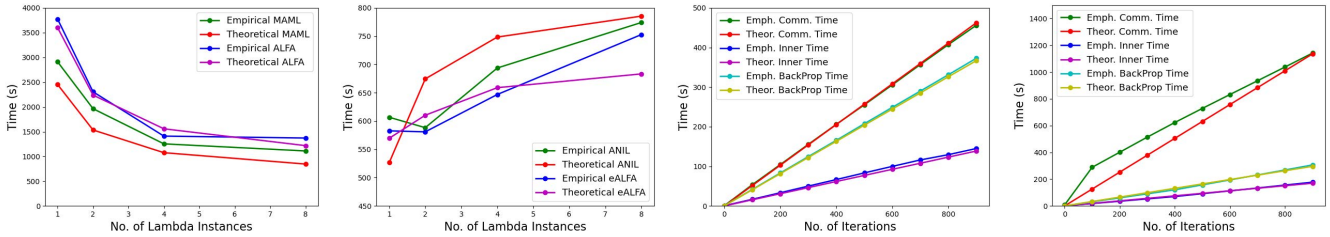
Based on the empirical study in Section 5, we devise an analytical model that captures the cost/performance trade-off for multiple configurations of meta-learning architectures, both on bare-metal and cloud and empirically validate the output of our analytical model. Given a meta-learning algorithm A, for which the task size is 't', model backbone is 'm', let number of workers instantiated be 'w'. Furthermore, let 'I' be the number of iterations required for convergence. Assume that the training function on lambda is invoked 'x' times, where:

$$x = \frac{I}{\text{No. of epochs completed in 15 min}} \tag{3}$$

then, a generalized equation of the analytical model for a given meta-learning algorithm 'A' is given as:

$$\begin{aligned} FaaS(w) : \\ = x\Big[t_s(w) + C_F(w) + \frac{I}{x}\Big[\frac{MBS}{w}\Big(\frac{t}{B_{S3}} + AS * IL(w) + B_K(w)\Big) \\ + \frac{(3w-2)}{c_3}\frac{m}{B_{S3}} + L_{S3}\Big] + S_C(w)\Big] + c \end{aligned} \tag{4}$$

where $t_s(w)$ is the startup time of a lambda instance, $C_F(w)$ and $S_C(w)$ are the loading and saving model time, MBS is the meta batch size, 'AS' is the number of adaptation steps in the inner loop of



(a) MAML - ALFA. **(b) ANIL - eALFA.** **(c) MAML - over 1000 iters.** **(d) ALFA - over 1000 iters.**

Figure 5: Empirical vs Theoretical Time using Analytical Model on FC100 5w1s Dataset.

the meta-learning algorithm, IL(w) is the inner loop update time that varies across the different meta-learning algorithms. B_{S3}, L_{S3} is the bandwidth and latency observed on S3. Lastly, c is a constant and $B_K(w)$ is the time required for back-propagating gradients and updating model parameters after each iteration. Again, the equation for $B_K(w)$ varies across the meta-learning algorithms. Detailed equations for each of the respective algorithms is discussed in Appendix B.

The graphs in Figure 4 clearly depict the trade-off between training the architecture on bare-metal and serverless, and the threshold beyond which we do not derive the benefits of training on serverless owing to the communication overhead. We now discuss the derivation of the communication cost for each worker.

Symbol	Configurations	Values
$t_s(w)$	w=1,2,4,8,16	(1.2 ± 0.1)s
B_{S3}	Amazon S_3 Bandwidth	(65 ± 7)MB/s
L_{S3}	Amazon S_3 Latency	(8 ± 2)×10^{-2} s
MBS	Meta-batch-size	4, 16, 32
AS	Adaptation Steps	1, 3, 5
I	Number of iterations	1k, 10k, 30k

Table 2: Constants for the analytical model

Workers on FaaS do not have any communication channel amongst themselves on invoking the lambda function, thus rendering it infeasible to accumulate gradients learned on each instance using conventional methods. We address this issue by making use of a disk-based object storage device such as S3 which enables read and write operations of the gradients. However our analytical model can easily be extended to other storage systems as well. The communication works as follows: (1) Each instance stores the gradients/current state in a temporary file and uploads it to S3. (2) One worker iterates over all the temporary files and merges them into a single file. (3) All other workers, except for the worker that has already read the file, read the final merged file from the storage system. (4) Lastly, the model parameters in each of the workers are updated with the latest aggregated parameters.

Thus, the equation for the communication time is given by:

$$\text{comm\_time(w)} = (3w - 2)\frac{t}{B_{S3}}\frac{I}{c_s} \tag{5}$$

where, c_s is a scaling factor that varies across the meta-learning algorithms. However the empirical results may vary depending on the size of the input dataset. And hence, we incorporate the scaling factor in the above equation for 1) number of workers; 2) dataset size.

Using suggested constants from Table 2, the analytical model approximates quite well to our empirical results as seen from the graphs in Figure 5. Figures 5a and 5b depict how the analytical model scales on FC100 dataset across an increasing number of instances on the 4 meta-learning algorithms. We observe in Figures 5c and 5d that the analytical model also scales well over increasing iterations for the *communication, inner-loop and back-propagation* time. Thus, our analytical model is generic across gradient based meta-learning architectures and provides useful insights into training cost incurred with scale.

7 CONCLUSIONS AND FUTURE WORK

The hierarchical nature of gradient-based meta-learning architectures enables them to scale well on a serverless setup. Limited memory and duration of serverless instances, render such architectures well suited for re-training meta-learning architectures on a few shots of data. This leads to a significant reduction in the overall training time (speed-up of 5X for large datasets such as Mini-Imagenet. Refer Table 3). eALFA achieves a speed-up of 9x on serverless, as compared to the original ALFA on a bare-metal serial setup. Our analytical model is largely generic and can be adapted to gradient-based meta-learning architectures with minor variations. As a part of the future work, we plan to improve upon the compute efficiency of meta-learning architectures, thus directly impacting their scalability, training time and cost incurred on the cloud.

Table 3: Speedup observed on Mini-Imagenet and CUBirds200 on FaaS. 'w' denotes the number of instances.

Dataset	MAML	ANIL	ALFA	eALFA
Mini-Imagenet	4.96x (16w)	2.16x (8w)	4.93x (16x)	9.14x (8w)
CUBirds200	5.68x (16w)	1.79x (8w)	5.04x (16x)	9.39x (8w)

REFERENCES

[1] 2022. Online data transfer and migration - AWS DataSync. https://aws.amazon.com/datasync/
[2] Amazon. 2021. Serverless on AWS. http://aws.amazon.com/serverless/. [Online; accessed 17-Dec-2021].

[3] Sungyong Baik, Myungsub Choi, Janghoon Choi, Heewon Kim, and Kyoung Mu Lee. 2020. Meta-Learning with Adaptive Hyperparameters. *CoRR* abs/2011.00209 (2020). arXiv:2011.00209 https://arxiv.org/abs/2011.00209

[4] Harkirat Singh Behl, Atilim Günes Baydin, and Philip H. S. Torr. 2019. Alpha MAML: Adaptive Model-Agnostic Meta-Learning. *CoRR* abs/1905.07435 (2019). arXiv:1905.07435 http://arxiv.org/abs/1905.07435

[5] Jan Bollenbacher, Florian Soulier, Beate Rhein, and Laurenz Wiskott. 2020. Investigating Parallelization of MAML. In *Discovery Science*, Annalisa Appice, Grigorios Tsoumakas, Yannis Manolopoulos, and Stan Matwin (Eds.). Springer International Publishing, Cham, 294–306.

[6] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. 2019. A closer look at few-shot classification. *arXiv preprint arXiv:1904.04232* (2019).

[7] Gonzalo Fernandez Dionis and Patricia C Mosser. 2022. The Structure of the Financial System: Implications for Macroprudential Stress Testing. *Handbook of Financial Stress Testing* (2022), 353.

[8] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *CoRR* abs/1703.03400 (2017). arXiv:1703.03400 http://arxiv.org/abs/1703.03400

[9] Li Fu, Xiaoxiao Li, Libo Zi, Zhengchen Zhang, Youzheng Wu, Xiaodong He, and Bowen Zhou. 2020. Incremental learning for end-to-end automatic speech recognition. *arXiv preprint arXiv:2005.04288* (2020).

[10] Google. 2021. Google Cloud. https://cloud.google.com/gcp/. [Online; accessed 17-Dec-2021].

[11] Andreas Grafberger, Mohak Chadha, Anshul Jindal, Jianfeng Gu, and Michael Gerndt. 2021. FedLess: Secure and Scalable Federated Learning Using Serverless Computing. *arXiv preprint arXiv:2111.03396* (2021).

[12] Christian Gross, Mariusz Jarmuzek, and Cosimo Pancaro. 2021. Macro-stress testing dividend income. Evidence from euro area banks. *Economics Letters* 201 (2021), 109763.

[13] Marco Gross, Jer^ome Henry, and Elena Rancoita. 2022. Macrofinancial Stress Test Scenario Design—for Banks and Beyond. *Handbook of Financial Stress Testing* (2022), 77.

[14] Gene Kitamura and Christopher Deible. 2020. Retraining an open-source pneumothorax detecting machine learning algorithm for improved performance to medical images. *Clinical imaging* 61 (2020), 15–19.

[15] Gregory R. Koch. 2015. Siamese Neural Networks for One-Shot Image Recognition. In *ICML Deep Learning Workshop*, Vol. 2.

[16] Shruti Kunde, Amey Pandit, Mayank Mishra, and Rekha Singhal. 2021. Distributed Training for Accelerating Metalearning Algorithms. In *Proceedings of the International Workshop on Big Data in Emergent Distributed Environments* (Virtual Event, China) *(BiDEDE '21)*. Association for Computing Machinery, New York, NY, USA, Article 2, 6 pages. https://doi.org/10.1145/3460866.3461773

[17] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. 2015. Human-level concept learning through probabilistic program induction. *Science* 350, 6266 (2015), 1332–1338. https://doi.org/10.1126/science.aab3050 arXiv:https://www.science.org/doi/pdf/10.1126/science.aab3050

[18] 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.

[19] Microsoft. 2021. Azure. https://azure.microsoft.com/en-us/. [Online; accessed 17-Dec-2021].

[20] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. 2017. Meta-Learning with Temporal Convolutions. *CoRR* abs/1707.03141 (2017). arXiv:1707.03141 http://arxiv.org/abs/1707.03141

[21] Ari S. Morcos, Maithra Raghu, and Samy Bengio. 2018. Insights on representational similarity in neural networks with canonical correlation. arXiv:1806.05759 [stat.ML]

[22] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, William Paul, Michael I. Jordan, and Ion Stoica. 2017. Ray: A Distributed Framework for Emerging AI Applications. *CoRR* abs/1712.05889 (2017). arXiv:1712.05889 http://arxiv.org/abs/1712.05889

[23] Tsendsuren Munkhdalai and Hong Yu. 2017. Meta Networks. *CoRR* abs/1703.00837 (2017). arXiv:1703.00837 http://arxiv.org/abs/1703.00837

[24] Dianwen Ng, Xiang Lan, Melissa Min-Szu Yao, Wing P Chan, and Mengling Feng. 2021. Federated learning: a collaborative effort to achieve better medical imaging models for individual sites that have small labelled datasets. *Quantitative Imaging in Medicine and Surgery* 11, 2 (2021), 852.

[25] Alex Nichol, Joshua Achiam, and John Schulman. 2018. On First-Order Meta-Learning Algorithms. *CoRR* abs/1803.02999 (2018). arXiv:1803.02999 http://arxiv.org/abs/1803.02999

[26] Alex Nichol, Joshua Achiam, and John Schulman. 2018. On First-Order Meta-Learning Algorithms. *CoRR* abs/1803.02999 (2018). arXiv:1803.02999 http://arxiv.org/abs/1803.02999

[27] Boris N. Oreshkin, Pau Rodríguez López, and Alexandre Lacoste. 2018. TADAM: Task dependent adaptive metric for improved few-shot learning. *CoRR* abs/1805.10123 (2018). arXiv:1805.10123 http://arxiv.org/abs/1805.10123

[28] Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. 2019. Rapid Learning or Feature Reuse? Towards Understanding the Effectiveness of MAML. *CoRR* abs/1909.09157 (2019). arXiv:1909.09157 http://arxiv.org/abs/1909.09157

[29] Sachin Ravi and H. Larochelle. 2017. Optimization as a Model for Few-Shot Learning. In *ICLR*.

[30] Marc Sánchez-Artigas and Pablo Gimeno Sarroca. 2021. Experience Paper: Towards enhancing cost efficiency in serverless machine learning training. In *Proceedings of the 22nd International Middleware Conference*. 210–222.

[31] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. 2016. Meta-Learning with Memory-Augmented Neural Networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48* (New York, NY, USA) *(ICML'16)*. JMLR.org, 1842–1850.

[32] Vishwas M Shetty and Metilda Sagaya Mary NJ. 2020. Improving the performance of transformer based low resource speech recognition for indian languages. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 8279–8283.

[33] Shaohuai Shi, Xianhao Zhou, Shutao Song, Xingyao Wang, Zilin Zhu, Xue Huang, Xinan Jiang, Feihu Zhou, Zhenyu Guo, Liqiang Xie, et al. 2021. Towards scalable distributed training of deep learning on public cloud clusters. *Proceedings of Machine Learning and Systems* 3 (2021).

[34] Jake Snell, Kevin Swersky, and Richard S. Zemel. 2017. Prototypical Networks for Few-shot Learning. *CoRR* abs/1703.05175 (2017). arXiv:1703.05175 http://arxiv.org/abs/1703.05175

[35] Oriol Vinyals, Charles Blundell, Timothy P. Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. 2016. Matching Networks for One Shot Learning. *CoRR* abs/1606.04080 (2016). arXiv:1606.04080 http://arxiv.org/abs/1606.04080

[36] Johannes von Oswald, Dominic Zhao, Seijin Kobayashi, Simon Schug, Massimo Caccia, Nicolas Zucchet, and João Sacramento. 2021. Learning where to learn: Gradient sparsity in meta and continual learning. *CoRR* abs/2110.14402 (2021). arXiv:2110.14402 https://arxiv.org/abs/2110.14402

[37] Risto Vuorio, Shao-Hua Sun, Hexiang Hu, and Joseph J. Lim. 2019. Multimodal Model-Agnostic Meta-Learning via Task-Aware Modulation. *CoRR* abs/1910.13616 (2019). arXiv:1910.13616 http://arxiv.org/abs/1910.13616

[38] Marcel Wagenländer, Luo Mai, Guo Li, and Peter Pietzuch. 2020. Spotnik: Designing Distributed Machine Learning for Transient Cloud Resources. In *12th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 20)*.

[39] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. 2011. *The Caltech-UCSD Birds-200-2011 Dataset.* Technical Report CNS-TR-2011-001. California Institute of Technology.

A PRELIMINARIES

Meta-learning is often used in few-shot learning paradigm to efficiently learn on unseen tasks given very few samples during training. To be specific, given an input data $\mathcal{D}$, the model converges well on test task $(x_i, y_i)_{i=1}^{K}$, where K is a small number. Thus, given a training data $\mathcal{D}_{\text{train}} = (x_i, y_i)_{i=1}^{S}$, the goal of the meta-learning algorithm $\mathcal{A}$ is to estimate the optimal initialization for parameters θ of the model f such that $\mathcal{L}(y, f(x;\theta))$, i.e., loss is minimized on the unseen test set $\mathcal{D}_{\text{test}} = (x_i, y_i)_{i=1}^{Q}$. In meta-learning literature, normally, the test and train set are sampled from the same distribution: $\mathcal{T}_i \sim p(\mathcal{T})$. However, we also consider the cross-domain scenario where test and train set are sampled from different distributions: $\mathcal{T}_i \sim p_{\text{train}}(\mathcal{T}); \mathcal{T}_j \sim p_{\text{test}}(\mathcal{T})$. In meta-learning, we consider the scenario of N-way K-shot learning. Here, N means the number of classification classes, and K are the number of samples per class. Thus, we can write $\mathcal{D}_{\text{train}} = \{(x_n^k, y_n^k)_{n=1}^{N} | k = 1, 2, 3, ...K\}$.

A.1 MAML

Model-agnostic Meta Learner (MAML) [8] tries to find a good initialization point for the model parameters. MAML works on the intuition that *a good starting point results in learning on new tasks efficiently using very few samples.* For each gradient update step, parameters are updated as follows:

$$\theta_{\mathcal{T}_i} = \theta_{\mathcal{T}_i} - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}^{\mathcal{D}_{\text{train}}}(f_\theta) \tag{6}$$

Whereas, in outer loop, MAML updates the initialization parameters of the model to an optimal point which results in fast convergence of loss to global minima. The meta-loss is defined as:

$$\mathcal{L}_{\text{meta}}(\theta) = \sum_{b=1}^{B} \mathcal{L}_{T_b}^{\mathcal{D}_{\text{test}}}(f_{\theta_b}(\theta)) \tag{7}$$

Using this loss the outer loop is updated as:

$$\theta = \theta - \eta \nabla_\theta \mathcal{L}_{\text{meta}}(\theta) \tag{8}$$

where, B (also known as the *metabatchsize*) is the number of tasks model is trained on in the current iteration and η is the meta learning rate of the outer loop. In the distributed training setup on FaaS, each task or a set of tasks in the meta-batch B, is executed on a worker. A copy of the model is trained on each worker. This is the inner loop of MAML. We achieve an acceleration in training time using serverless, as the tasks in the meta-batch are distributed and hence executed simultaneously on each worker.

A.2 ANIL

The ANIL [28] algorithm is a simplified version of MAML, where the parameter updates of the inner loop are considered redundant.

Mathematically, let $\theta = (\theta_1, \theta_2, \theta_3, ..., \theta_l)$ be the (meta-initialization) parameters for the l layers of the network. Then,

$$\theta_{\mathcal{T}_i} = (\theta_1, ..., \theta_l)_{\mathcal{T}_i} - \alpha \nabla_\theta \mathcal{L}_{\theta_{l_{\mathcal{T}_i}}}^{\mathcal{D}_{\text{train}}}(f_\theta) \tag{9}$$

i.e. only the final layer gets the inner loop updates. The meta-loss, and outer-loop gradient update are same as in Equation 7, and Equation 8. ANIL is distinct to the freezing experiments, where the inner loop is removed only at the inference time. All the models are updated in the outer loop. Because of it's similarity with MAML, ANIL can be easily adapted to the serverless architecture, similar to MAML. The tasks in the meta-batch are distributed across multiple workers. However since computation in the inner-loop is already optimized (only the head layer is updated), we do not expect to see a very high speed-up in the distributed serverless setup for ANIL.

A.3 ALFA

ALFA [3] is a gradient-based meta-learning algorithm, that focuses on adaptive learning of hyperparameters for fast adaptation, i.e., inner-loop optimization. It incorporates an l_2 regularization term $\frac{\lambda}{2}||\theta||_2$ to the loss function $\mathcal{L}_{\mathcal{T}_i}$ to address potential overfitting. Thus, the MAML inner loop Equation 6 differs from ALFA inner loop equation as follows:

$$\begin{aligned} \theta_{\mathcal{T}_i} &= \theta_{\mathcal{T}_i} - \alpha(\nabla_\theta \mathcal{L}_{\theta_{l_{\mathcal{T}_i}}}^{\mathcal{D}_{\text{train}}}(f_\theta) + \lambda \theta_{\mathcal{T}_i}) \\ &= \beta \theta_{\mathcal{T}_i} - \alpha \nabla_\theta \mathcal{L}_{\theta_{l_{\mathcal{T}_i}}}^{\mathcal{D}_{\text{train}}}(f_\theta) \end{aligned} \tag{10}$$

where $\beta = 1 - \alpha\lambda$. The adaptation process depends on the hyperparameters in the inner-loop update equation, which are scalar constants of learning rate α and regularization hyperparameter β. The dynamic variables $\alpha_{i,j}$ and $\beta_{i,j}$ are generated using a learnable neural network g_ϕ in every inner loop iteration. The parameters of g_ϕ are updated in the outer loop after every iteration.

B ANALYTICAL MODEL FOR META-LEARNING ARCHITECTURES

The execution time in the analytical model varies slightly for each of the gradient-based meta-learning algorithms. Equation 4 is largely generic, with changes to $IL(w)$ and $B_K(w)$. The communication time is architecture agnostic, and dependent on the size of input task, model, and number of instances invoked. We now turn our attention to the cost (in dollars) incurred during training. The cost is calculated as a scaling factor of execution time and memory utilized on lambda instances. Additional cost is computed as a function of the read/write operations on S3 and size of data transferred from S3 to EFS using DataSync [4]. Thus our analytical model is also capable of providing an accurate estimate of the cost incurred across a range of datasets, algorithms, and training iterations. We now outline the analytical model equations for each of the meta-learning architectures.

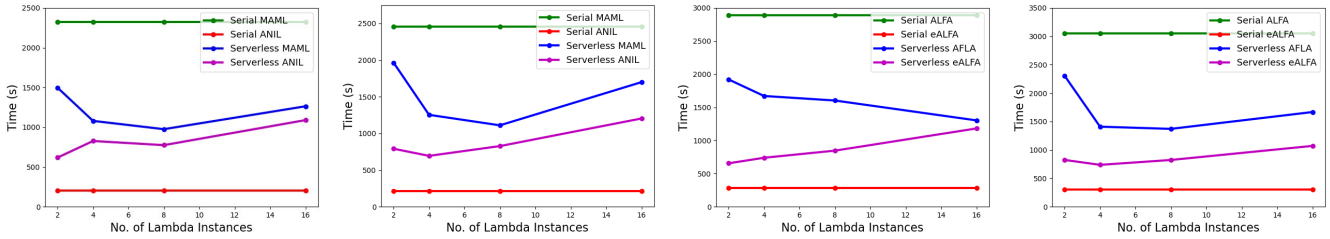
$$\begin{aligned} MAML/ANIL(w) := x\Big[t_s(w) + 4mtc_1 + \frac{I}{x}\Big[\frac{MBS}{w}\Big(\frac{t}{B_{S3}} \\ + c_2(1+AS)\frac{t}{2} + \frac{(3w-2)}{c_3}\frac{m}{B_{S3}}\Big) + L_{S3}\Big]\Big] + c \end{aligned} \tag{11}$$

The values of the constants c_1, c_2, c_3 vary depending on the dataset and model used. For ANIL, the only change in the Equation 11 will be the decrease in the number of parameters (m') model is being trained on. This reduced model size results in the observed speedup of ANIL over MAML.

$$\begin{aligned} ALFA/eALFA(w) := x\Big[t_s(w) + 4m_0tc_1 + \frac{I}{x}\Big[\frac{MBS}{w}\Big(\frac{t}{B_{S3}} \\ + c_2(1+AS)\frac{t}{2} + \frac{(3w-2)}{c_3}\frac{m}{B_{S3}} + c_4(m+N+6N^2)AS\Big) + L_{S3}\Big]\Big] + c \end{aligned} \tag{12}$$

[4] Datasync is a fast and efficient method to transfer data across various file systems on AWS [1].

Figure 6: Additional results - Performance of meta-learning algorithms on FaaS and Serial Setup.



(a) Omniglot 5w1s.MAML-ANIL **(b) FC100 5w1s.MAML-ANIL** **(c) Omniglot 5w1s. ALFA-eALFA** **(d) FC100 5w1s. ALFA-eALFA**

where, m_0 is the size of the neural network g_ϕ and N represents the number of layers in the base model. ALFA updates model parameters in inner loop using dynamically generated hyper-parameters for each layer. Thus we need to incorporate the additional parameters in our model. In the outer loop, only the g_ϕ parameters are updated. The base model parameters are not updated. eALFA differs from ALFA in Equation 12 only in the inner loop, where we only update the weights of the head layer.

Using the suggested constants from Table 2, the analytical model approximates quite well to our empirical results as seen from the figures in Figure 5. Figures 5a and 5b depict how the analytical model scales on FC100 dataset across increasing number of instances on the 4 meta-learning algorithms. We observe in Figures 5c and 5d that the analytical model also scales well over increasing iterations for the *communication, inner-loop and back-propagation* time. Thus, our analytical model is generic across gradient based meta-learning architectures and provides useful insights in training cost incurred with scale.

C RESULTS

C.1 Performance acceleration on serverless

We present additional results (Figure 6) for 5ways-1shot, depicting the performance of MAML, ANIL, ALFA and eALFA on serverless and bare-metal. The discussion of the results is found in Section 5.1.