2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid) | 978-1-7281-9586-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/CCGrid51090.2021.00096

High Performance Serverless Architecture for Deep Learning Workflows

Dheeraj Chahal
TCS Research
Mumbai, India
d.chahal@tcs.com

Manju Ramesh
TCS Research
Mumbai, India
manju.ramesh1@tcs.com

Ravi Ojha
TCS Research
Mumbai, India
ravi.ojha@tcs.com

Rekha Singhal
TCS Research
Mumbai, India
rekha.singhal@tcs.com

***Abstract*—Serverless architecture is a rapidly growing paradigm for deploying deep learning applications performing ephemeral computing and serving bursty workloads. Serverless architecture promises automatic scaling and cost efficiency for inferencing deep learning models while minimizing the operational logic. However, serverless computing is stateless with constraints on local resources. Hence, deploying complex deep learning applications containing large size models, frameworks, and libraries is a challenge.**

In this work, we discuss a methodology and architecture for migrating deep vision algorithms and model based applications to a serverless computing platform. We have tested our methodology using AWS infrastructure (AWS Lambda, Provisioned Concurrency, VPC endpoint, S3 and EFS) to mitigate the challenges in deploying composition of APIs containing large deep learning models and frameworks. We evaluate the performance and cost of our architecture for a real-life enterprise application used for document processing.

***Index Terms*—Serverless, FaaS, Cloud, AI**

I. Introduction

Artifical Intelligence (AI) model training is compute intensive, time consuming, and expensive that requires from few hours to days and hence, performed offline using high performance machines. However, model inference is performed in real-time and requires high throughput with low latency to meet the quality of service (QoS) requirements. Bursty workloads for inference necessitate the use of scalable architecture. Over-provisioning the computation resources for such workloads to ensure high performance, results in higher costs and low overall resource utilization. This has been one of the reasons for migrating the applications from an on-premise deployment to cloud which offers elasticity by scaling-up and scale-down the resources.

Serverless computing has become a platform of choice for migrating event driven applications to cloud. It has been applied to the area of machine learning and deep learning with partial success [1] [2]. Although training AI model is still a challenge in serverless computing, but prediction and inference has shown promising results.

Serverless architecture provides highly scalable and cost effective deployment of event triggered applications. Function as a Service (FaaS) platform, for serverless computing, offers thousands of compute cores to the developers for achieving autoscaling with ease. Moreover, the pay-per-use cost model in FaaS platforms provides cost-efficiency as well. For burst workloads, the elasticity of the function is offloaded to the FaaS platform where the function can scale up and down and no cost is incurred to the user when the system is idle. FaaS platform is available from all leading cloud providers such as AWS Lambda offered from Amazon [1], Microsoft Azure functions [2], Google cloud functions [3], and IBM OpenWhisk [4].

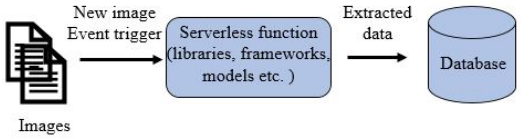


Fig. 1: Image processing flow

Despite the several advantages available with the serverless computing, developers run into difficulty while migrating large applications due to stateless nature of cloud functions and restrictions on local resources. Accommodating large application APIs, packages, libraries, and large deep learning models for inference in the constraints of small local storage and memory of FaaS is challenging.

Hence, one of the challenging use cases is the deployment of deep vision algorithm based applications on cloud using a serverless computing platform [3] for extracting information from images and PDF documents (Figure 1). Such applications consist of an orchestration pipeline of large deep learning models, interspersed with pre-processing and post-processing functions. These applications serve batch workloads where high throughput is expected. The highly scalable and cost-effective pay-per-use model in serverless architecture is suitable for such workloads.

In this work, we present a methodology and high performance architecture for deploying complex vision algorithm based applications using serverless architecture. Succinctly, our contributions are as follows:

1) We discuss the challenges and solutions for migrating large deep learning model based applications from the

[1]https://aws.amazon.com/lambda/
[2]https://azure.microsoft.com/en-in/services/functions/
[3]https://cloud.google.com/functions
[4]https://www.ibm.com/in-en/cloud/functions

978-1-7281-9586-5/21/$31.00 ©2021 IEEE
DOI 10.1109/CCGrid51090.2021.00096

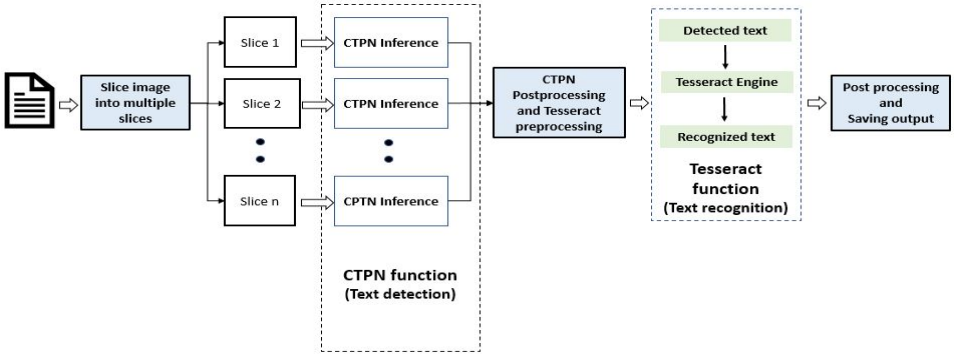


Fig. 2: On-premise deployment of the Deep Reader application's OCR API. Each slice is processed in a separate thread inside the CTPN function. The number of parallel threads depends on the server configuration.

on-premise deployment to the serverless architecture on cloud.

2) We propose an architecture for deploying such applications using multiple cloud services such as storage and file systems.
3) We study performance and cost incurred in deploying deep learning model based applications on the serverless architecture platform. Furthermore, a comparison vis-à-vis on-premise deployment is presented.

Although we use the well known Amazon Web Service (AWS) serverless architecture platform called AWS Lambda, the proposed methodology and architecture can be implemented using cloud-provider-specific APIs and services.

The rest of the paper is structured as follows. In Section II, we discuss related work. The image processing application used in this work is presented in Section III. Our migration methodology is discussed in Section IV followed by the architecture for deploying this application on FaaS platform is explained in Section V. The experimental setup and experiment results are explained in Section VI and VII respectively.

II. Related Work

In one of the pioneer works, SPEC RG Cloud working group introduced a reference architecture and ecosystem for FaaS platforms [4]. In the latest technical report published by the group, a review of 89 serverless use cases is presented [5]. Each use case in this study is reviewed based on characteristics such execution pattern, use of external services, workflow coordination, etc. A detailed survey on serverless platforms from industry, academia, and open-source projects is presented in [6]. Baldini et al. [3] discussed technical challenges and open problems in their study on serverless computing. FaaS challenges such as overhead, scheduling, performance isolation and prediction are discussed in [7].

In [8], experiment design and toolkit is proposed to evaluate the FaaS platform. Also, based on the results of the experiments, applicability of FaaS platform to different applications is discussed in this work.

Use of serverless computing for deep neural networks has been explored successfully [9]. In a work similar to ours [10],

Fig. 3: Image processing by the OCR API. Input image in sliced in multiple parts for improving the accuracy of object detection and recognition by the CTPN model and Tesseract engine.

suitability of serverless architecture for inferencing large neural networks is evaluated. Although deep learning models used in this work vary in size up to 100 MB, these models can still easily fit in the local storage available in the FaaS. A study involving Migration of monolithic document processing system used in FinTech, to serverless architecture is presented in [11]. However, our work is focused on deploying models bigger than the ephemeral storage.

A recent work elaborates methods and techniques to migrate large libraries and frameworks of AI applications [12] to FaaS platform. Zheng et al. conducted a study on serverless platform for cost-effective and SLO aware machine learning inference [13]. We extend this work further for large models that can not fit in the FaaS temporary storage.

To the best of our knowledge, our work is among the first efforts to explore performance and cost in migrating on-premise vision algorithm based applications containing the pipeline of APIs and large deep learning models, to cloud using serverless architecture features such as provisioned concurrency and elastic file system. We believe that this work will provide directions to address the challenges faced in deploying such applications on cloud.

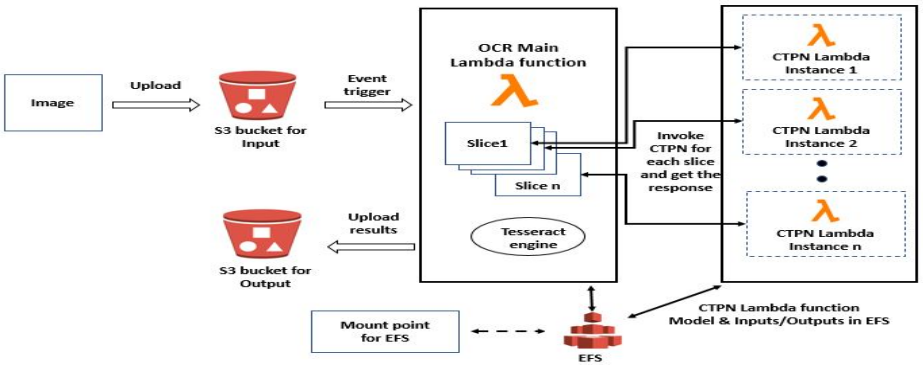


Fig. 4: Deep Reader workflow on serverless architecture using AWS Lamda, S3 and EFS

III. Our Application and Workflow

In this section, we discuss the design of Deep Reader [14], our in-house application that we migrated to FaaS platform. Deep Reader is an invoice processing application built with the composition of a number of APIs. These APIs implement state-of-the-art vision algorithm and contain pre-trained large deep learning models. The application logic performs the data extraction as well as transformations. There are multiple workflows in the applications to process images from different classes. Each workflow consists of one or more application APIs. The important APIs in the application includes the Optical Character Recognition (OCR) API for reading characters, words and sentences along with co-ordinates, extractor API for detecting semantic data types of the words, table detection API for identifying the tables in the images or pdf document and last but not least, is the relationship extractor API for identifying the relationship among the different entities in the document.

The OCR API extracts the text and its corresponding coordinates from the image or document (Figure 2). Thus, we get the words, sentences, text blocks, and their location in the image. The two major steps are text detection (to find out where the text is present), and text recognition (to identify the words). The Connectionist Text Proposal Network (CTPN) [15] model is used for text detection and the Tesseract [16] engine is used for recognition.

For better accuracy in CTPN inference, the image is split into multiple slices with overlapping regions (Figure 3). Each slice is given as the input to the CTPN model, and it provides bounding box coordinates of the regions where the text is present. The overlapping boxes are combined and the batch of unique polygons containing text is then sent to Tesseract engine for recognizing the text.

Sentences and Text blocks mapped to their coordinates is the final output of the OCR API. It is returned as JavaScript Object Notation (JSON) and is consumed by other APIs.

Among all the APIs, OCR API is the most frequently used API and essentially used in all the workflows. Also, it is compute intensive and consumes more than the 70% of the total image processing time and hence, a major focus of this work.

In our previous work [17], we have shown a primitive architecture for deploying large models on serverless computing platform. We extend this work further by introducing provisioned concurrency feature for low latency and parallel execution of CTPN functions. We also enhanced the architecture to support end-to-end deployment of the OCR API.

IV. Migration methodology

In this section, we discuss optimizations and procedures for deploying the Deep Reader application on cloud. We use AWS serverless platform called AWS Lambda in conjunction with Simple Storage Service (S3) and Elastic File System (EFS). The AWS Lambda has only 500 MB of /tmp directory storage and only 250 MB available for package deployment. However, our applications uses large deep learning models, frameworks, and libraries such as CTPN (270 MB) and TensorFlow (220MB). In order to overcome the resource constraints in the FaaS platform, we have applied some methods and optimizations as discussed below.

A. Clipping TensorFlow and Python libraries

The TensorFlow framework used by Deep Reader is approximately 220 MB is size. We trim some of the components from TensorFlow (e.g. Tensorboard) before deploying it on Lambda function. Also, some of the libraries that are not required by our application are eliminated. With this exercise, we see around 50% reduction in the size of code that needs to be migrated to the Lambda function. Additionally, both Tesseract engine and and OpenCV [5] models are loaded in the Lambda layers for sharing across multiple Lambda functions.

B. Elastic File System for storage

In our implementation, we load the CTPN model from EFS instead of loading them from S3. The intermediary files generated and used by the Lambda instances are also stored in EFS for low latency. Although S3 is cheaper in comparison to the EFS, but latter has Max I/O performance modes for higher

[5]https://opencv.org/

write speeds and throughput. The default throughput of 3GB/s is available in EFS however at a higher price as compared to S3.

C. Using provisioned Lambda resources

The recently introduced provisioned concurrency feature for Lambda keeps functions warm and ready to respond with consistent latency in few milliseconds [6]. The cold start problem in Lambda functions can be mitigated by provisioning the Lambda resources. Based on the expected workload, we provision resources in advance to keep the Lambda instances warm for the incoming requests. However, provisioning Lambda resources results in additional cost. Moreover, provisioning time shall be chosen judiciously to minimize resource idle time.

V. Deployment Architecture

The deployment architecture for the Deep Reader workflow on serverless platform is as shown in Figure 4. An event is triggered when the image is uploaded in the S3 bucket. The event notification from S3 results in image processing by the OCR Lambda function containing the OCR API. The OCR function slices each image into multiple parts and stores them in EFS. These image slices generated by OCR function are consumed by an equivalent number of CTPN Lambda instances invoked by the OCR function. The CTPN lambda function process slices in parallel and the generated files are returned back to the OCR for further processing by Tesseract. Tesseract is an OCR engine, coded in C and C++. Tesseract and its prerequisites need to be compiled using the GNU C compiler compatible with AWS Lambda. A Lambda Layer is created with the compiled binaries and configuration files, this is then imported in the OCR main Lambda function.

All AWS services used in the architecture are placed in one Virtual Private Cloud (VPC). We use the Gateway Type VPC endpoint instead of the NAT Gateway to route the S3 traffic. Files shared among OCR and CTPN Lambda functions are put in EFS while all other files are saved in /tmp of respective Lambdas. The final output data generated by the OCR API is stored in a separate S3 bucket. We upload input data and save output data in S3 due to the data accessibility of S3 objects over internet and cross region replication feature which is a constraint in EFS.

VI. Experimental Setup

The workload we use in our experiments consist of invoice images containing hand written text, tables, logos, etc. Our local deployment is on Intel Xeon 1359v2@2.45 GhZ with 48 cores (Hyper-Threading on) configured with 64 GB of memory. We use the local file system to read images and store output results.

For experiments with serverless architecture, images to be processed are uploaded on pre-configured buckets of Amazon S3 in batches of different sizes. An event trigger was setup on S3 to invoke the Lambda function as soon as an image is uploaded. We have analyzed batches in our study to understand the impact on the cost and performance at high concurrency. The AWS Lambda instance we use can be configured with memory upto 3GB and 512MB of disk space. The number of cores allocated on the AWS Lambda instances is directly proportional to the amount of memory allocated with a maximum of two virtual CPUs being allocated at 3GB. Each experiment is run minimum three times and average of the data is calculated for the final results. To achieve concurrency in execution of AWS Lambda instances, all jobs are submitted in less than 3 seconds.

In our experiments, we have configured OCR function to generate nine slices of each input image to be processed in parallel by an equivalent number of CTPN functions as discussed in Section V.

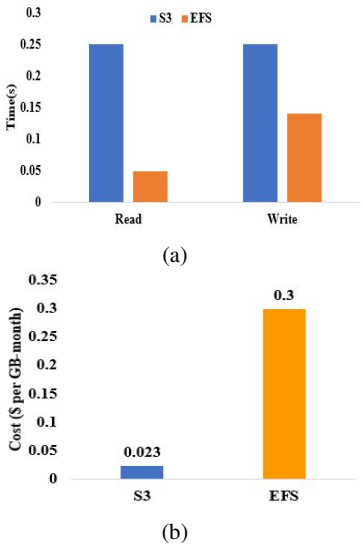


Fig. 5: AWS S3 and EFS latency and cost comparison

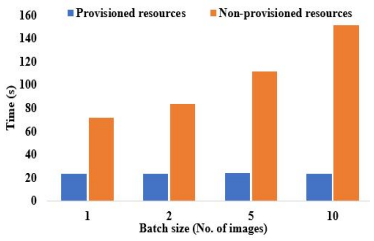
VII. Experimental Evaluation

In this section, we discuss the experiments conducted to evaluate the performance and the associated cost of the architecture that we have proposed. In all our experiments, the end-to-end image processing time includes the time to load an image in S3, transfer from S3 to the Lambda function, time in the Lambda function and loading results back in S3.

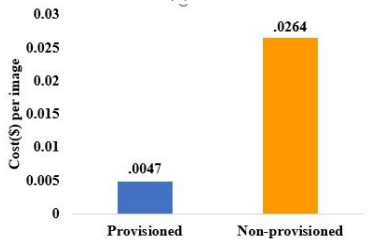
A. S3 vs EFS cost evaluation

This experiment evaluates the read and write latency in S3 and EFS that we use to store input/output data and intermediary files. As shown in Figure 5a, the read and write latency of S3 is constant while the EFS performs almost 5X better in read and 2X better in write operations compared to S3. However cost of EFS, as shown in Figure 5b, is approximately 10X of S3.

[6]https://aws.amazon.com/blogs/aws/new-provisioned-concurrency-for-lambda-functions/



(a)



(b)

Fig. 6: Comparison of (a) Total execution time and (b) cost using provisioned and non-provisioned resources on AWS Lambda

Hence, EFS is good for storing small intermediary files that are shared by multiple Lambda functions and require low latency.

B. Provisioned concurrency vs cold start performance and cost

In this experiment, we evaluate the execution time and the cost incurred with the provisioned concurrency which results in consistent start-up latency and without provisioned concurrency in Lambda which then results in cold start. As shown in Figure 6a, The total time for processing the images does not change with provisioned concurrency. We also observe that for a given workload, the image processing time observed in non-provisioned resources is more than that in provisioned resources. Along with this, latency increases in non-provisioned Lambda instances as the workload increases. As we increase the workload,the number of instances invoked in both the provisioned and non-provisioned concurrency experiments increase proportionally as discussed in Section V. However, high scalability without fluctuations in latency results in constant processing time in provisioned concurrency experiments.

Although the provisioning concurrency cost per unit time per GB data is more as compared to non-provisioned Lambda functions for same memory allocated, the effective cost using provisioned concurrency is less than non-provisioned Lambda functions as shown in Figure 6b. This is due to the longer execution time in cold start CTPN Lambda functions.

C. On-premise vs Lambda scaling and performance

In this experiment, we study the scaling by increasing the batch size. As shown in Figure 7, for smaller batch sizes,

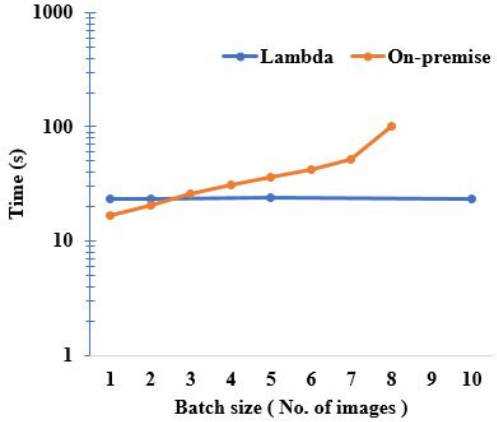


Fig. 7: Comparison of on-premise and AWS Lambda performance with scaling

the processing time increases linearly but for larger batch sizes, it increases exponentially in an on-premise environment. However, processing time on the Lambda does not change with the batch size. This is due to the fact that mutithreading, for executing slices of each image in the OCR API, scales well till all the cores on the on-premise server are saturated. With large batch sizes, cores on the local server are saturated, resulting in a bottleneck and higher latency. However, highly scalable serverless architecture allows spawning new Lambda instances in proportion to the workload as discussed in Section V.

As shown in the graph, Lambda instances with limited resources (2 cores for each function) result in poor performance in comparison to the bare metal on-premise machine for small batch sizes; but as we increase the batch size, overhead due to local execution is more than the resource-constraint but scalable Lambda instances.

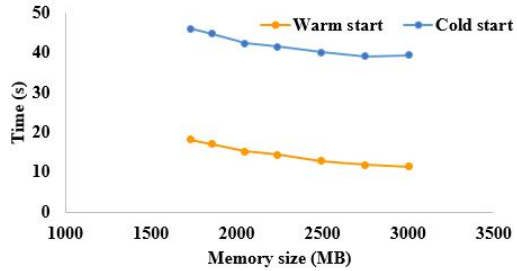


Fig. 8: Effect of memory size configured on processing time due to cold start and warm start of CTPN Lambda functions

D. Performance and cost with varying memory configuration

In this experiment, we study the impact of memory size configuration in each Lambda instance on cold start and warm start. As shown in Figure 8, with increase in memory size, per image processing time decreases in the CTPN Lambda function due to the availability of more compute power.

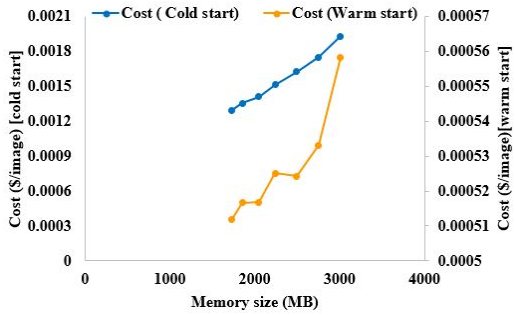


Fig. 9: Effect of memory size configured on cost with cold start and warm start of image processing in CTPN Lambda functions

TABLE I: AWS Lambda memory cost

Memory (MB)	Price per 100ms
128	$0.0000002083
512	$0.0000008333
1024	$0.0000016667
1536	$0.0000025000
2048	$0.0000033333
3008	$0.0000048958

Figure 9 shows the increase in the cost of processing in the CTPN Lambda function with memory. A comparison of Figure 8 and 9 shows that the performance gain is insignificant at higher memory sizes whereas the increase in the cost is significant. This is due to the fact that the cost of FaaS is derived not only from the execution time but also from the memory allocated (Table I). Similar behavior is observed in the end-to-end processing of an image by the OCR and CTPN Lambda functions.

As observed in Figure 10a, cumulative processing time on an image (OCR and CTPN Lambda function) decreases with more memory allocation to Lambda functions. Although, the rate of decrease in execution time becomes insignificant at higher memory sizes (above 2 GB) but the cost of processing increases at the same rate (Figure 10b).

This study shows that choosing Lambda configurations judiciously might result in saving cost significantly without compromising the performance. This experiment can help practitioners in understanding the performance and the cost trade-off.

VIII. Conclusion

We proposed a methodology for migrating an on-premise application containing large deep learning models and multiple workflows to a serverless architecture on cloud. Also, we proposed a deployment architecture using multiple cloud services such as S3, EFS, VPC etc. Additionally, we conducted performance and cost evaluation for the proposed architecture using recently introduced provisioned concurrency feature by AWS. Our experimental results show that, when used

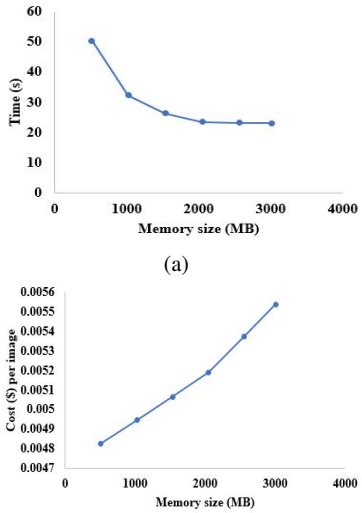


Fig. 10: Effect of memory size configured on (a) processing time (b) cost of processing an image by OCR API

judiciously, serverless computing delivers high performance with substantial cost reduction.

The workload characterization for addressing the cold start problem in FaaS, is a work in progress. To enable seamless interaction between different API's of our application, we plan to enable queues provided by AWS as Simple Queue Service (SQS). Queues help in overcoming the challenge of tracking messages that do not get processed either due to FaaS or networking issues between different components. SQS provides high scalability by controlling throttling of messages as per demand.

In our current implementation, the output of the OCR pipeline containing text blocks and sentences, is generated as JavaScript Object Notation (JSON). These JSON objects are saved in EFS and S3 storage. In future, we plan to integrate Amazon kev-value NoSQL database called DynamoDB, in our workflow for low latency.

References

[1] L. Feng, P. Kudva, D. Silva, and J. Hu, "Exploring serverless computing for neural network training," *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pp. 334–341, 2018.

[2] J. Carreira, P. Fonseca, A. Tumanov, A. Zhang, and R. Katz, "Cirrus: a serverless framework for end-to-end ml workflows," 11 2019, pp. 13–24.

[3] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski, and P. Suter, *Serverless Computing: Current Trends and Open Problems*. Singapore: Springer Singapore, 2017, pp. 1–20.

[4] E. van Eyk, J. Grohmann, S. Eismann, A. Bauer, L. Versluis, L. Toader, N. Schmitt, N. Herbst, C. L. Abad, and A. Iosup, "The spec-rg reference architecture for faas: From microservices and containers to serverless platforms," *IEEE Internet Computing*, vol. 23, no. 6, pp. 7–18, 2019.

[5] S. Eismann, J. Scheuner, E. van Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. L. Abad, and A. Iosup, "A review of serverless use cases and their characteristics," *arXiv preprint arXiv:2008.11110*, 2020.

[6] J. Scheuner and P. Leitner, "Function-as-a-service performance evaluation: A multivocal literature review," *Journal of Systems and Software*, vol. 170, p. 110708, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121220301527

[7] E. Van Eyk, A. Iosup, C. L. Abad, J. Grohmann, and S. Eismann, "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*, 2018, pp. 21–24.

[8] J. Kuhlenkamp, S. Werner, M. C. Borges, D. Ernst, and D. Wenzel, "Benchmarking elasticity of faas platforms as a foundation for objective-driven design of serverless applications," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, ser. SAC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1576–1585. [Online]. Available: https://doi.org/10.1145/3341105.3373948

[9] A. Bhattacharjee, A. D. Chhokra, Z. Kang, H. Sun, A. Gokhale, and G. Karsai, "Barista: Efficient and scalable serverless serving system for deep learning prediction services," in *2019 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2019, pp. 23–33.

[10] V. Ishakian, V. Muthusamy, and A. Slominski, "Serving deep learning models in a serverless platform," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2018, pp. 257–262.

[11] A. Goli, O. Hajihassani, H. Khazaei, O. Ardakanian, M. Rashidi, and T. Dauphinee, "Migrating from monolithic to serverless: A fintech case study," in *Companion of the ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 20–25. [Online]. Available: https://doi.org/10.1145/3375555.3384380

[12] A. Christidis, S. Moschoyiannis, C.-H. Hsu, and R. Davies, "Enabling serverless deployment of large-scale ai workloads," *IEEE Access*, vol. 8, pp. 70 150–70 161, 2020.

[13] C. Zhang, M. Yu, W. Wang, and F. Yan, "Mark: Exploiting cloud services for cost-effective, slo-aware machine learning inference serving," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. Renton, WA: USENIX Association, Jul. 2019, pp. 1049–1062. [Online]. Available: https://www.usenix.org/conference/atc19/presentation/zhang-chengliang

[14] D. Vishwanath, R. Rahul, G. Sehgal, S. , A. Chowdhury, M. Sharma, L. Vig, G. Shroff, and A. Srinivasan, *Deep Reader: Information Extraction from Document Images via Relation Extraction and Natural Language*, 06 2019, pp. 186–201.

[15] Z. Tian, W. Huang, H. Tong, P. He, and Y. Qiao, "Detecting text in natural image with connectionist text proposal network," vol. 9912, 10 2016, pp. 56–72.

[16] R. Smith, "An overview of the tesseract ocr engine," in *Ninth international conference on document analysis and recognition (ICDAR 2007)*, vol. 2. IEEE, 2007, pp. 629–633.

[17] D. Chahal, R. Ojha, M. Ramesh, and R. Singhal, "Migrating large deep learning models to serverless architecture," in *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2020, pp. 111–116.