When Serverless Computing Meets Different Degrees of Customization for DNN Inference

Moohyun Song
mhsong@kookmin.ac.kr
Computer Science. Kookmin Univ.
Seoul, South Korea

Yoonseo Hur
yoonseo@kookmin.ac.kr
Computer Science. Kookmin Univ.
Seoul, South Korea

Kyungyong Lee
leeky@kookmin.ac.kr
Computer Science. Kookmin Univ.
Seoul, South Korea

ABSTRACT

Serverless computing provides a method to develop application services without the burden of run-time execution environment management overhead. Since the initial offerings of serverless computing using function-as-a-service (FaaS), other variants of execution environments have been proposed, such as a special-purpose FaaS (SPF) for deep neural network (DNN) inference and a serverless container service (SCS) for general web applications. This paper qualitatively summarizes the characteristics of a general-purpose FaaS (GPF), SPF, and SCS from the perspective of customizability when setting up execution environments. To judge whether various serverless computing environments can be feasible solutions for an interactive DNN model inference application, we conduct extensive experiments and conclude that there are rooms for performance improvement serverless DNN inference, and allowing a custom environment setup can make the serverless computing platform for an interactive DNN application.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**.

KEYWORDS

serverless computing, dnn inference

ACM Reference Format:
Moohyun Song, Yoonseo Hur, and Kyungyong Lee. 2023. When Serverless Computing Meets Different Degrees of Customization for DNN Inference. In *9th International Workshop on Serverless Computing (WoSC '23), December 11–15, 2023, Bologna, Italy*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3631295.3631400

1 INTRODUCTION

Cloud computing flexibly provides computing resources with on-demand pricing. Users can easily build highly available application services using diverse features, such as auto-scaling and load balancers, with cloud instances, which are referred to as infrastructure-as-a-service (IaaS). Since the wide adoption of the initial IaaS model, cloud computing services have been evolving in the direction of hiding complex service operations so that users can focus on core application development [14].

A higher degree of computing service abstraction becomes feasible with the development of diverse fully managed services, such as database, messaging, object storage, and FaaS, allowing users to implement custom source codes. Combining FaaS and fully managed services delivers serverless computing, freeing users from cumbersome resource operations [8]. Although the introduction of serverless computing enables an innovative method of application development in the cloud, some limitations can hinder the adoption of serverless computing in a wide range of applications [5].

Current public serverless computing providers do not support direct communications among function runtimes, and a function runtime does not have permanent storage that mandates external storage services [2, 12]. Users have a limited capability to set up runtime environments due to memory size, and most other settings are configured by service vendors [17]. Due to this limited capability, current serverless computing applications are restricted to a simple back-end application server, parallel jobs, and cloud service orchestrations [3, 5, 9, 10, 15]. To enhance serverless computing to a broader set of applications, specially designed serverless computing platforms have also been proposed (lithops [11] and Pywren [7]), but they are not yet commercially available.

When developing serverless applications, the degree of customization available to set up runtime environments can have a significant influence on overall performance. As the serverless computing eco-systems have evolved, various runtime platforms and services have been introduced. The FaaS platform is most widely used for current serverless application development, and the major cloud vendors provide the service (AWS Lambda, Google, Microsoft and IBM Cloud Functions). Recently, an SPF platform was also provided. For instance, AWS released the SageMaker Serverless Inference service which was specially designed for DNN inference tasks. In addition, SCS to serve web applications has been provided. GCP Cloud Run and AWS AppRunner services are typical examples.

Distinct serverless computing environments offer various customization degrees for runtimes, but the performance differences among different execution runtimes are not adequately explored yet. To solve this issue, we compare the GPF, SPF, and SCS environments for DNN inference, highlighting their unique applications. Our observations then guide the potential enhancements and challenges for interactive DNN applications in serverless computing.

From our experimental results, we have derived a variety of insights as follows.

- For most DNN models, SPF shows better performance than GPF mainly due to its model serving engine provided by default.

WoSC '23, December 11–15, 2023, Bologna, Italy

ACM ISBN 979-8-4007-0455-0/23/12…$15.00
https://doi.org/10.1145/3631295.3631400

- By comparing end-to-end response times, we identify the API endpoint protocol has a higher impact to impact the overall latency than the inference time.
- In SCS, using more instances offers superior performance with lower cost than increasing CPU cores per instance.
- Cold-start can take over 10s of seconds to load a DNN model and serving libraries, suggesting necessity for further research.

2 SERVERLESS EXECUTION ENVIRONMENTS

The execution environment of serverless computing is managed and offered by service providers. Initially, GPF was the only option to develop serverless applications. As serverless application scenarios become diverse, SPF and SCS have been introduced.

2.1 General Purpose FaaS

When using FaaS, although the detailed operating mechanism can slightly differ, users first register the custom source code written in various programming languages, such as Java, Python, JavaScript, Go, Ruby, or C#. Registered functions operate on an event-driven basis, and users can register a variety of fully managed event sources. In cases where a function must be invoked based on user requests, an HTTP endpoint can be registered. During registration, users must determine various configuration settings, such as the maximum memory size that the function runtime can use, and the maximum time that the source code can execute. Representative services include AWS Lambda, Azure Functions, IBM Cloud Functions, and Google Cloud Functions.

2.2 Special Purpose FaaS for DNN Inference

SPF is a new type of FaaS designed for specific tasks. The recently released AWS SageMaker Serverless Inference service is a representative example. The service offers a specially constructed inference environment that allows the deployment and scaling of a machine learning model serving environment on the AWS Lambda FaaS platform. Figure 1 illustrates the internal operating mechanism of the AWS SageMaker serverless inference service, which mainly consists of an HTTP server and a container. The container encapsulates the user-defined inference code, libraries, and other dependencies. Upon initialization, the container loads the model, making it ready to run when an inference request arrives. The HTTP server, acting as a handler, manages incoming inference requests. It predominantly interfaces with the model via RESTful API, and API calls are addressed by the HTTP server. When the HTTP server receives an inference request, it passes the request via REST or gRPC to an internal ML framework like TFServing that resides in the same container, calls the model, and returns the results to the user. Besides SageMaker serverless inference, other examples of SPF include IBM Cloud Functions integrated with Watson services, which allow for more effective execution of specific tasks such as natural language processing and data analytic.

2.3 Serverless Container Service

SCS allow developers to deploy and run user's container images which can contain custom applications written in any desired languages using libraries without the burden of managing instances.

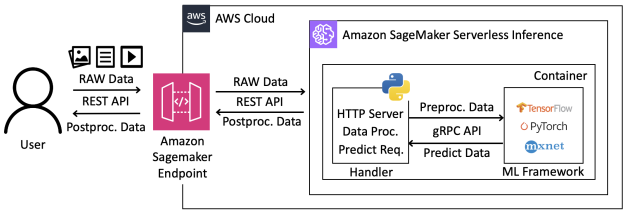


Figure 1: An example architecture of special-purpose FaaS for DNN inference

Since the introduction of AWS Fargate as an SCS, FaaS-style container services, such as GCP Cloud Run and AWS AppRunner, have been offered. The critical difference between them is that Cloud Run and AppRunner automatically scales up and down rapidly as requests come in, which is similar to FaaS. In contrast Fargate provides a method to set more general in options based on the underlying container runtime service. Additionally, similar services are provided by AWS App Runner, Azure Container Instances (ACI) and IBM Cloud Code Engine.

3 CUSTOMIZING SERVERLESS COMPUTING RUNTIME FOR DNN INFERENCE

Different serverless execution environments have unique configurations when running DNN inference tasks, and we highlight them qualitatively from the perspective of the runtime setup, scaling, and API endpoint protocol. We characterize various services based on the representative services in each category: AWS Lambda for GPF, AWS SageMaker Serverless Inference for SPF, and GCP Cloud Run for SCS.

3.1 Runtime Environment

In general, users register only the function when using FaaS. However, to support cases when the user code and library become complex and large, the service supports a user's custom container image as a function run-time. For an SPF for DNN inference, a service vendor provides an optimized model server platform, such as TFServing, PyTorch, and MXNet, and users do not need to prepare their own image. Users can add additional data processing steps before and after an inference task. In the SCS, users must prepare their own container images to execute. Furthermore, there is no public endpoint handler implementation, and users must manually provide an external API endpoint.

3.2 Runtime Configuration and Scaling Setup

To set up GPF and SPF runtime configurations, users generally specify the maximum memory size for runtime, and the CPU resources are allocated by a service provider proportionally to the memory size. Users have more flexibility when setting SCS for GCP Cloud Run. One can determine the concurrency that specifies the number of requests a single instance can manage simultaneously. Users can also specify the minimum and maximum number of instances, ensuring that a certain number of instances can run regardless of the degree of requests. Additionally, users can define the number of cores per instance, with a default value of one. The billing is calculated by considering the configured values.

	Runtime	Scaling Configuration	Network Protocol
General FaaS	★★	★	★
Special FaaS	★	★	★★
Serverless Container	★★★	★★★	★★★

Table 1: The relative degree of customization (the more stars, the more customization possible) for different serverless execution environments.

3.3 API Endpoint Protocol

When serving an interactive application, the HTTP REST API is widely used to transmit and receive data. Further, REST is an architectural style for designing networked applications over HTTP, and its principal advantage is its remarkable flexibility. In inference tasks, components of the REST API include the endpoint, which is a URL for the client to access the server, the HTTP method indicating the type of operation the client wishes to perform, and the request and response.

As an alternative to the HTTP REST API, gRPC is a modern and high-performance RPC framework, capable of running in any environment. Furthermore, gRPC communicates over HTTP/2, supporting multiple bidirectional streams on a single TCP connection. It uses protocol buffers, a serialization structure, on top of the HTTP/2 layer, which is lighter than JSON-based communication and eliminates the need for separate parsing, thereby accelerating communication. Thus, it offers the advantages of reduced communication time and the capability to handle increased traffic. The current GPF supports only the REST API and not the gRPC. The external endpoint of SPF supports only the REST API, but a user can use gRPC for an internal communication mechanism. SCS supports both gRPC and HTTP REST for the external end-point which provides the most capability.

Table 1 compares the degree of customization for various serverless execution environments. The number of stars in each cell represents the degree of customizability. For all criteria, the SCS allows users to have the most freedom when setting up the environment.

4 EVALUATION OF SERVERLESS EXECUTION ENVIRONMENT

This section focuses on the quantitative evaluation of the influence of customization on performance when serving a DNN model.

4.1 Experiment Setup and Workload

To evaluate the performance of the DNN model inference under various serverless execution environments, we used AWS Lambda, AWS SageMaker Serverless Inference and GCP CloudRun to represent GPF, SPF and SCS, respectively. For DNN inference workloads, we used MobileNetV1 [13] and InceptionV3 [16] as image classification models, YOLOV5 [6] as an object detection model, and BERT [4] for natural language processing. The input dataset for MobileNetV1 is a three-channel image with a pixel size of 224×224, and the InceptionV3 size is 299×299. For YOLOV5, we used the COCO dataset(640×640). For BERT, we used the IMDb movie review dataset for sentiment classification.

Unless otherwise noted, to mitigate the effects of a cold-start, we pre-warmed the execution environments by invoking inference tasks with the same configuration in each experiment. The GPF received JSON requests containing input data via REST through the Amazon API Gateway. Then these data were parsed by the AWS Lambda Handler and subsequently passed to the TensorFlow library [1] for inference. For the SPF, JSON containing protobuf input data is sent via REST to the AWS SageMaker Inference Endpoint. The handler parses the JSON and delivers the protobuf data to TFServing using gRPC, residing in the same container. SCS used the HTTP/2 endpoint in GCP Cloud Run, enabling direct connections to the container via gRPC to transmit protobuf data. This container was designed to receive requests through a Python-written gRPC server, process protobuf data, and make predictions using TensorFlow.

Model	GFLOPS	Model Size	Input Size		Output Size	
			gRPC	REST	gRPC	REST
MobileNetV1	1.15	18	0.574	3.014	0.0040	0.0268
InceptionV3	11.5	97	1.023	5.524	0.0040	0.0265
YOLOV5	16.5	28	4.688	24.547	8.172	63.5932
BERT	13.39	428	0.006	0.004	0.0001	0.0001

Table 2: Model and input/output dataset sizes (MB)

Model	Inference Time (Sec.)			
	Python		TFServing	
	Disabled	Enabled	Disabled	Enabled
MobileNetV1	0.082	0.085	0.037	0.03
Inceptionv3	0.357	0.203	0.187	0.148
YOLOV5	0.331	0.255	0.345	0.276
BERT	1.830	1.294	1.827	1.246

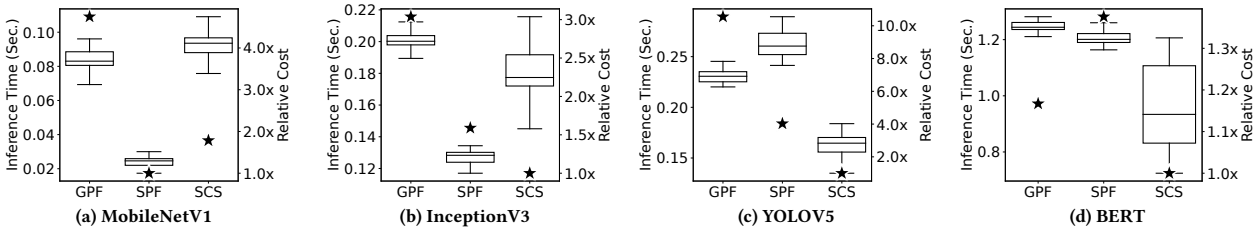
Table 3: The impact of AVX512 instruction support for DNN inference tasks. Disabled/Enabled means AVX512 feature status.

4.2 DNN Inference Performance

Figure 2 presents the inference time of warm-start cases with various settings for distinct models that are shown in the sub-figures. In the experiments, 40 concurrent requests were made. Each platform was allocated 3GB of memory. For SCS, we set the minimum number of instances as 40 to ensure a warm start and configured it with 2 cores per instance, ensuring that it had the same core count as GPF and SPF. The primary vertical axis presents the inference time, and the values are presented using a box-whisker format. The secondary vertical axis displays the cost incurred to process a request. The cost was normalized to that of the minimum value in each configuration and the values were marked using star markers. The horizontal axis presents run-time environments for GPF, SPF, and SCS in order.

Two image classification models (Figures 2a and 2b) display similar patterns. SPF has the fastest inference time. A thorough investigation reveals that the optimized model serving platform provided by SPF contributed to improve inference time. The SPF supports TFServing and other inference engines by default. For SCS and GPF, users must manually install, optimize, and operate the inference engine, which can be challenging and cumbersome.

For YOLOV5 (Figure 2c) and BERT (Figure 2d), the SCS performs best. Our investigation reveals that the superior performance is owing to the newer generation CPUs that is offered by GCP Cloud



(a) MobileNetV1 (b) InceptionV3 (c) YOLOV5 (d) BERT

Figure 2: The inference time (primary Y-axis) to serve a single request and the corresponding cost (secondary Y-axis).

Run. The Skylake-SP Intel CPU, which is provided by SCS, supports AVX512 instructions with SIMD feature, and inference tasks could gain a significant performance improvement over FaaS. To see the impact of AVX512 instruction for inference, Table 3 shows the impact of AVX512 on inference performance. To measure the AVX512 impact in a controllable manner, we used a m5.large EC2 instance type which supports AVX512 and installed the same software packages as in the serverless execution environments. To test the performance without AVX512, we disabled the feature by setting a CPUID. We can observe that the larger models have a higher performance gain than smaller models. Regardless of the size, models have default overhead other than the core computation kernels, and smaller models (MobileNetV1 and InceptionV3) have higher overhead ratio than larger models. For smaller models, the performance gain from using TFServing over a custom implementation of inference service using Python web-server and TensorFlow on the GPF is noticeable. However, for larger models, since the computation takes most of the time, using TFServing did not result in a significant performance gain. This result demonstrates that the performance can be significantly influenced by the characteristics of the hardware of the runtime, and specially-designed hardware for inference keeps released. For example, recent Intel Xeon Max CPU is equipped with High Bandwidth Memory with Advanced Matrix Extensions (AMX) support to deliver much higher inference performance than previous generation CPUs. Thus, in a serverless environment where cloud vendors choose which hardware to provide, users should be able to understand the characteristics and utilize its features.

Regarding cost, the SCS showed the most cost-efficiency, and we expect that this is due to a higher degree of scaling policy customization and lower cost of the service. Please note that though we are running the minimum of 40 machines for GCP Cloud Run to measure warm-start time, only the processing time is included in the cost calculation. For YOLOV5, we can observe that the inference time of GPF is slightly faster than SPF, but the cost of GPF is higher than SPF. The reason is related to the additional overhead due to the extra data transfer overhead to and from the external object storage. The measured inference time is time for the inference only, but the overall cost is calculated by measuring a handler execution time which includes additional data preparation time. The additional network overhead is going to be further analyzed next.

Figure 3 compares the end-to-end response time to complete a client's inference task. The latency in Figure 2 includes the inference time only, and this figure includes network latency. One of the primary factors that affects network latency is the API endpoint protocol. The HTTP REST API is widely used, and the current AWS

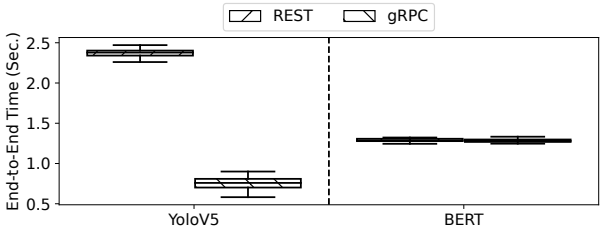


Figure 3: End-to-end response time for an inference task

Lambda (GPF) supports only the REST API, to which we had to stick. AWS SageMaker serverless inference (SPF) has more flexibility in choosing inference API endpoint protocol, and we used gRPC to pass input and output for an inference task. The horizontal axis presents two different models, YOLOV5 and BERT, which have distinct input and output sizes from an inference task, as presented in Table 2. In case of YOLOV5, which has a large input and output dataset that are not supported by AWS FaaS endpoint, we used an object storage service, S3, to pass the input/output to and from an user. For each model, we present two cases, one using REST API in a JSON format (GPF) and the other using gRPC in a Protobuf format (SPF). The size of the input and output datasets is expressed in Table 2. As revealed in the graph, the difference in response time between the gRPC and REST API protocols becomes noticeable when a large data set is required (YOLOV5). Compared to the inference time difference between runtimes, which is in the range of hundreds of milliseconds, the response time has a difference of about 2 second. This is because gRPC transmits and receives data through the Protobuf format, resulting in a smaller data size compared to REST (JSON). This experimental result indicates that, when managing a client's inference task, the network latency can take most of the response time, and the hardware performance of the serverless computing environment might not be a dominant factor when determining user experiences. From the context, a more efficient API endpoint should be natively provided for GPF.

4.3 Scalability

The internal scaling mechanism is different for FaaS and SCS. Figure 5 illustrates the performance of FaaS as the configured memory size changes. For comparison, we present two different models, a small MobileNetV1 model (Figure 4a) and a large BERT model (Figure 4b). The primary vertical axis indicates the inference time and the secondary vertical axis displays the relative cost normalized to when the configured memory size is 2GB. The horizontal axis

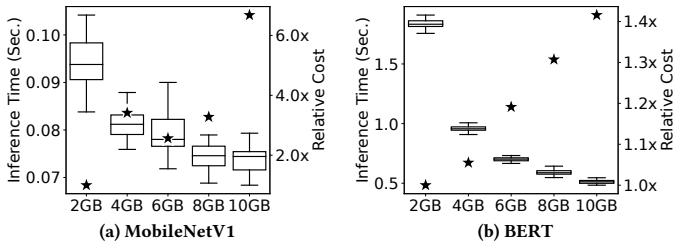


Figure 4: Inference time as the configured memory size changes for general purpose FaaS

presents the configured memory size of FaaS. Both models show improving performance as the memory size increases, agreeing with the advertisements by the service providers that the CPU capacity is allocated proportionally to the configured memory size. However, the performance improvement rate presents quite a difference. The smaller model exhibits a minor performance improvement even after adding more resources, resulting in a significant overall cost increase. The larger BERT has a slight increase in cost as more resources are allocated, but the performance improvement is also noticeable. In FaaS, to provide more compute capacity as more memory is allocated, the number of allocated CPU cores increases, and if the application is not using the CPU cores in parallel, the performance gain from allocating more memory can be negligible. From this finding, we can conclude that the optimal FaaS environment configuration is crucial to obtaining the best performance and cost efficiency, and it can considerably vary greatly when serving various DNN models.

Figure 5 presents the performance variations of SCS as we change the number of instances, the maximum number of requests that an instance can process concurrently, and the number of assigned CPU cores per instance change. We show a small and a large model case, and other models show a similar pattern. The numbers on the horizontal axis separated by the underscores represent the value in order. In each configuration, we present the inference time on the left and the end-to-end response time on the right. The primary vertical axis displays the latency and the secondary vertical axis reveals the relative cost normalized to the minimal cost in each configuration. In addition, SCS has a rather complex pricing mechanism, and the vendor does not provide per-request billing. Therefore, we manually calculated the cost following the billing criteria of the service.

In the experiments, we issued 48 requests at once and measured the processing time while using 8GB of RAM. From the figures, we can discover that the inference time does not make much difference across different settings. However, the end-to-end latency and cost have noticeable variations for different settings. When handling requests, the number of instances and CPU cores determines the cost. We discovered that packing more requests to a CPU core (the case of 6_8_8 and 6_8_2) to save cost does not affect the inference time, but the end-to-end response time becomes longer due to the request queuing from the SCS engine. With the same degree of the number of instances times the number of CPU cores, in the cases of 12_4_4 and 24_2_2, using a larger number of instances resulted in better performance than allocating more CPU cores in an instance with a lower cost. However, using more instances can negatively

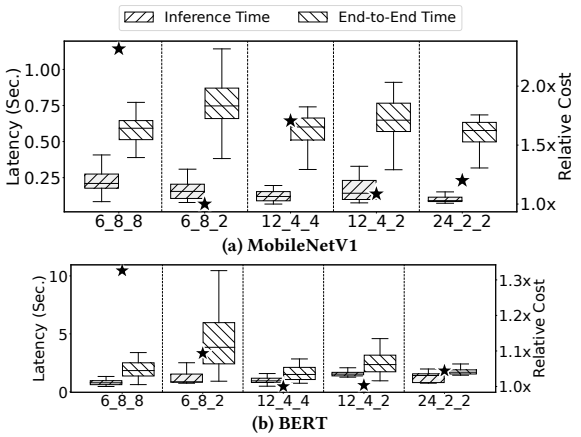


Figure 5: Inference latency and response time as the configured number of instances, concurrency, and the number of CPU cores per instance changes for SCS

affect the performance when a cold-start occurs, which we discuss next. Despite the general trend for such a performance change, different models show slightly different patterns, and further research should be conducted to suggest optimal configurations for SCS runtimes.

When using serverless computing, users should be aware of the cold-start. So far, the experimental result is based on the warm-start cases, and Figure 6 shows the cold start time for the small and large DNN models of MobileNetV1 and BERT, respectively. In the experiments, we issued 40 requests at once and measured the cold start time, and then calculated the average value. For each platform, we configured it to use 3GB of RAM. The horizontal axis indicates various serverless runtimes and the primary vertical axis displays the latency. In some cases, multiple requests cannot be completed when a cold-start occurs, and the number is presented on the secondary vertical axis with a star marker. The lower part of a stacked bar represents the model loading time, and the upper part represents the time for loading containers and the necessary libraries to serve a model. To generate a cold-start, we issue 40 concurrent executions in newly created environments. As presented in the figure, it takes significant time for loading a DNN model, container image, and necessary libraries to serve an inference service. In the case of GPF, the container load time is smaller than SPF and SCS because a separate HTTP or gRPC server library is not included in the image. In the BERT model, we can see that the container load time has increased due to the large size of the model, as presented in Table 2. Taking into account the inference latency and network latency to serve a user's request, the latency when cold-start occurs increases to over 10 seconds, which can severely affect user experiences. Furthermore, the SCS of GCP Cloud Run had many request drops in the cold start, where about half of the requests could not be completed. Despite the acceptable response time in case of warm-start, the cold-start latency is far from supporting an interactive DNN inference application, and further research and engineering effort should be devoted to overcome this issue.

From the comprehensive experiments of serving the DNN model using various serverless execution environments, we could uncovered the following.

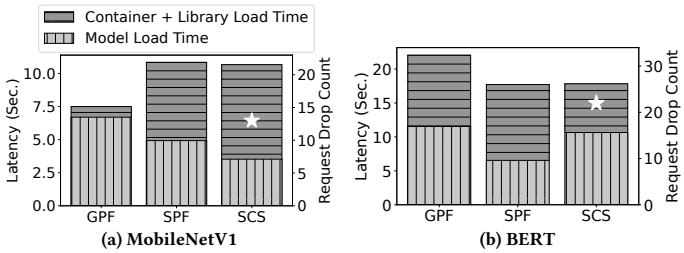


Figure 6: The DNN inference latency when cold-start happens. Compared to the warm-start time, the inference cold-start latency is significant.

- An optimized inference service engine of an SPF can perform better than a GPF with lower overhead
- The API network protocol has a significant influence on the end-to-end response time to manage an inference tasks, and more degree of freedom should be given to users for better performance
- The complex configuration space of SCS offers a new research challenge to provide an optimal environment for various DNN models.
- Despite the affordable inference time of DNN inference on a warm-start, the cold-start latency (over tens of seconds) is not affordable for an interactive application, and further research specific to enhancing the DNN inference cold-start time is necessary to widely adopt serverless computing widely for DNN inference.

5 CONCLUSION AND FUTURE WORK DIRECTION

Since the introduction of serverless computing and a GPF as a runtime execution environment, other variants of serverless execution environments have been developed. An SPF for DNN inference provides an optimized environment for a workload, an SCS provides more flexibility in setting up the runtime. Using the various setups, we thoroughly analyzed the inference performance of the DNN model. The findings suggests the feasibility of using serverless computing as a model serving framework, but it has limitations as an interactive inference service. Further research and analysis can be done in the direction of applying a special purpose library, such as DeepSpeed Inference [18], on a serverless environment. Performance analysis of a large model, such as Large Language Model, on a constrained serverless environment should be conducted.

6 ACKNOWLEDGMENTS

This work is supported by the National Research Foundation of Korea (NRF) or Institute of Information & communications Technology Planning & Evaluation (IITP) Grant funded by the Korean Government (MSIT) : NRF-2020R1A2C1102544, NRF-2022R1A5A7000765, and RS-2022-00144309 (SW StarLab).

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: a system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*. 265–283.

[2] Jaeghang Choi and Kyungyong Lee. 2020. Evaluation of Network File System as a Shared Data Storage in Serverless Computing. In *Proceedings of the 2020 Sixth International Workshop on Serverless Computing* (Delft, Netherlands) *(WoSC'20)*. Association for Computing Machinery, New York, NY, USA, 25–30. https://doi.org/10.1145/3429880.3430096

[3] U. Choi and K. Lee. 2022. Dense or Sparse : Elastic SPMM Implementation for Optimal Big-Data Processing. *IEEE Transactions on Big Data* 01 (aug 2022), 1–17. https://doi.org/10.1109/TBDATA.2022.3199197

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL 2019*.

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

[6] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, Kalen Michael, TaoXie, Jiacong Fang, imyhxy, Lorna, Zeng Yifu, Colin Wong, Abhiram V, Diego Montes, Zhiqiang Wang, Cristi Fati, Jebastin Nadar, Laughing, UnglvKitDe, Victor Sonck, tkianai, yxNONG, Piotr Skalski, Adam Hogan, Dhruv Nair, Max Strobel, and Mrinal Jain. 2022. *ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation*. https://doi.org/10.5281/zenodo.7347926

[7] Eric Jonas, Qifan Pu, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. 2017. Occupy the Cloud: Distributed Computing for the 99%. In *Proceedings of the 2017 Symposium on Cloud Computing* (Santa Clara, California) *(SoCC '17)*. ACM, New York, NY, USA, 445–451. https://doi.org/10.1145/3127479.3128601

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

[9] J. Kim and K. Lee. 2019. FunctionBench: A Suite of Workloads for Serverless Cloud Function Service. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. https://doi.org/10.1109/CLOUD.2019.00091

[10] Jeongchul Kim and Kyungyong Lee. 2019. Practical Cloud Workloads for Serverless FaaS. In *Proceedings of the ACM Symposium on Cloud Computing* (Santa Cruz, CA, USA) *(SoCC '19)*. ACM, New York, NY, USA.

[11] Josep Sampé, Marc Sánchez-Artigas, Gil Vernik, Ido Yehekzel, and Pedro García-López. 2023. Outsourcing Data Processing Jobs With Lithops. *IEEE Transactions on Cloud Computing* 11, 1 (2023), 1026–1037. https://doi.org/10.1109/TCC.2021.3129000

[12] Marc Sánchez-Artigas and Germán T. Eizaguirre. 2022. A Seer Knows Best: Optimized Object Storage Shuffling for Serverless Analytics. In *Proceedings of the 23rd ACM/IFIP International Middleware Conference* (Quebec, QC, Canada) *(Middleware '22)*. Association for Computing Machinery, New York, NY, USA, 148–160. https://doi.org/10.1145/3528535.3565241

[13] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4510–4520. https://doi.org/10.1109/CVPR.2018.00474

[14] Johann Schleier-Smith, Vikram Sreekanti, Anurag Khandelwal, Joao Carreira, Neeraja J Yadwadkar, Raluca Ada Popa, Joseph E Gonzalez, Ion Stoica, and David A Patterson. 2021. What serverless computing is and should become: The next phase of cloud computing. *Commun. ACM* 64, 5 (2021), 76–84.

[15] M. Son and K. Lee. 2018. Distributed Matrix Multiplication Performance Estimator for Machine Learning Jobs in Cloud Computing. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, Vol. 00. 638–645. https://doi.org/10.1109/CLOUD.2018.00088

[16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2015. Rethinking the Inception Architecture for Computer Vision. *CoRR* abs/1512.00567 (2015). arXiv:1512.00567 http://arxiv.org/abs/1512.00567

[17] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2018. Peeking Behind the Curtains of Serverless Platforms. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 133–146. https://www.usenix.org/conference/atc18/presentation/wang-liang

[18] Reza Yazdani Aminabadi, Samyam Rajbhandari, Minjia Zhang, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Jeff Rasley, Shaden Smith, Olatunji Ruwase, and Yuxiong He. 2022. *DeepSpeed Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale*. Technical Report MSR-TR-2022-21. Microsoft. https://www.microsoft.com/en-us/research/publication/deepspeed-inference-enabling-efficient-inference-of-transformer-models-at-unprecedented-scale/