# All-You-Can-Inference : Serverless DNN Model Inference Suite

Subin Park*
subean@kookmin.ac.kr
Computer Science. Kookmin Univ.
Seoul, South Korea

Jaeghang Choi*
chl8273@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 becomes prevalent and is widely adopted for various applications. Deep learning inference tasks are appropriate to be deployed using a serverless architecture due to the nature of fluctuating task arrival events. When serving a Deep Neural Net (DNN) model in a serverless computing environment, there exist many performance optimization opportunities, including various hardware support, model graph optimization, hardware-agnostic model compilation, memory size and batch size configurations, and many others. Although the serverless computing frees users from cloud resource management overhead, it is still very challenging to find an optimal serverless DNN inference environment among a very large optimization opportunities for the configurations. In this work, we propose All-You-Can-Inference (AYCI), which helps users to find an optimally operating DNN inference in a publicly available serverless computing environment. We have built the proposed system as a service using various fully-managed cloud services and open-sourced the system to help DNN application developers to build an optimal serving environment. The prototype implementation and initial experiment result present the difficulty of finding an optimal DNN inference environment with the varying performance.

## CCS CONCEPTS

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

## KEYWORDS

serverless computing, dnn inference

**ACM Reference Format:**
Subin Park, Jaeghang Choi, and Kyungyong Lee. 2022. All-You-Can-Inference : Serverless DNN Model Inference Suite. In *Eighth International Workshop on Serverless Computing (WoSC '22), November 7, 2022, Quebec, QC, Canada.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3565382.3565878

## 1 INTRODUCTION

Cloud computing has become prevalent in diverse application scenarios. Since its introduction, it has evolved in the direction of hiding complex resource management and operation overheads with a higher degree of computing resource abstraction. To that end, the serverless computing frees users from the burden of server maintenance for high-availability, enabling them to focus on core application development. Diverse applications are deployed adopting the serverless architecture taking advantage of innate high-availability support from various fully-managed services.

DNN model inference tasks, which take trained DNN models to serve user requests, can take advantages of the elastic compute resource offering of serverless computing as the degree of user requests can vary greatly. However, there exist many challenges when setting up serverless DNN inference execution environments, such as limited local file storage, no direct communication among run-times [13], and rather unstable performance [19, 22]. Despite these challenges, the current public Function-as-a-Service (FaaS) and serverless computing provide numerous opportunities for the performance optimization of DNN inference tasks. These include the use of new types of hardware (e.g., ARM architecture server), diverse ephemeral and permanent file storage (object, in-memory key-value, relational databases), DNN model optimizations, and optimizing compute resource allocation [27].

The challenges and opportunities of running DNN inference using a serverless architecture co-exist, and it can be very difficult for ordinary application developers to understand the characteristics of up-to-date public FaaS offerings and apply them to their deployment. Furthermore, different DNN models can result in distinct performance patterns in a diverse setup [6, 24, 28], and it is barely generalizable. To expedite the adoption of a serverless architecture for various applications including DNN inference, the environment setup and finding optimally performing configurations should be effortless. To achieve this goal, we develop All-You-Can-Inference (AYCI) which provides a diverse set of DNN inference environments adopting a serverless architecture as a web-based service. The current implementation supports various hardware types (Intel and ARM cpus), DNN model optimization (TVM [8] and ONNX [10]), and development platforms (TensorFlow [1], PyTorch [23], and MXNet [7]). Users can also set FaaS memory size and batch size during model serving. Using the proposed system, users can submit their custom DNN models and receive thorough reports on how the models would behave with diverse configurations of FaaS.

During the prototype implementation, we tackled lots of technical challenges as the development pace of relevant technologies in the prototype is very fast, and it requires extensive efforts to make it work. To confirm the proposed system work correctly, we conducted thorough testing, and we could uncover quite drastic performance variations as the FaaS configuration and DNN inference environment change which are barely generalizable. The qualitative challenges in building diverse serverless DNN inference systems and the quantitative performance variations in the serverless inference environments demonstrate the necessity of the proposed system to provide easy-to-experiment environment.

In summary, the major contributions are as follows.

*Both authors contributed equally to this work.


*WoSC '22, November 7, 2022, Quebec, QC, Canada*

ACM ISBN 978-1-4503-9927-2/22/11…$15.00
https://doi.org/10.1145/3565382.3565878

- The development of various serverless DNN inference environments and sharing them publicly to help users understand workload performance characteristics on serverless environments[1].
- Uncovering new insights regarding the DNN inference performance variability through comprehensive experiments on serverless environments.

## 2 SERVERLESS COMPUTING FOR DNN MODEL INFERENCE

In a traditional cloud computing service model, users must decide target services and resource types to launch applications with the responsibility for operating the resources. Maintaining cloud resources in a highly available manner is not a trivial task which can be challenging for ordinary cloud system users. Serverless computing allows service operators to deploy applications without the burden of maintaining a highly available system by using diverse fully-managed cloud services [13, 25], including FaaS. Since its inception, the usage of serverless computing has been expanding, and the DNN model inference can well be deployed using the serverless computing architecture [15]. Unlike DNN model training which generally requires significant amount of computing power for a relatively longer duration, the amount of computing power required for DNN model inference is largely dependent on the model size and the burstiness of users' requests. The innate auto-scaling support of FaaS is vital to handle users' dynamic inference requests. Furthermore, different from DNN model training that generally process a batch of input dataset to achieve higher throughput, a model inference task is generally executed in a single input to shorten the latency. In general, using GPU is more beneficial for batch processing. Given that GPU devices are not supported in the current FaaS yet, serving a DNN model inference task using CPU with a small batch size can become a reasonable choice.

However, few limitations imposed in the FaaS might hinder the adoption of DNN model inference. First of all, DNN model serving requires a large size of library packages to perform inference tasks. For example, serving a model built using TensorFlow, PyTorch, and MXNet requires about 3GB, 4.4GB and 1.8GB of storage for each library, except a model. The large package size can negatively impact the overall processing throughput especially when a cold-start happens [26, 27] because a FaaS run-time should load all the necessary packages from the file storage. Furthermore, most public FaaS vendors provide limited local file storage of variable sizes which is not persistent across different executions, and we cannot make sure the packages can be stored locally in a function.

The evolution of FaaS and serverless computing provides numerous opportunities for serving a DNN model. Newly announced FaaS file storage services [4, 9, 12, 14] can be a solution to store large DNN packages. The cold-start time can be mitigated by preparing a function run-time proactively [5]. In addition to widely-used Intel architecture, new hardware is also supported from public FaaS vendors, such as ARM architecture hardware from AWS Graviton. In setting up a function run-time, the memory size can greatly impact the overall performance, and the variation is rather non-deterministic [18, 19, 22]. Batching can help improve serving throughput while satisfying users' QoS [2]. The flexibility of FaaS programming models allow the adoption of DNN model graph optimizer and hardware compilers, such as ONNX [10] and TVM [8].

[1]https://github.com/ddps-lab/lambda-optimize-serving

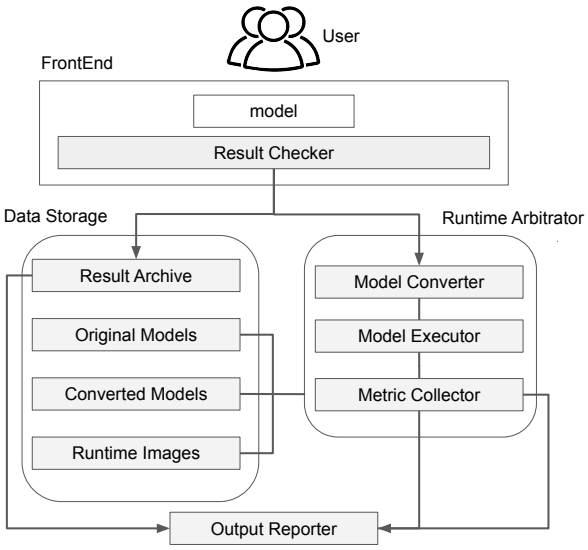



**Figure 1: Major components of All-You-Can-Inference**

Despite the opportunities of the performance improvement, it is very challenging for DNN application developers to try all the possible deployment scenarios provided by a serverless computing. In the literature, many research works have thoroughly evaluated and compared public FaaS vendors [21, 27]. Few works have provided an open-source workload suite that can help researchers and developers to evaluate public FaaS vendors [16, 17, 29]. However, to the best of the authors' knowledge, there are no prior work that can help developers to deploy DNN model serving workloads using the serverless computing that provides a large configuration space.

## 3 ALL-YOU-CAN-INFERENCE : SERVERLESS DNN INFERENCE SUITE

In this work, we propose All-You-Can-Inference, which is a publicly available DNN inference environment suite on a serverless computing environment. The goal of AYCI is to help DNN application developers estimate the performance of inference tasks on various configurations of FaaS so that users can build an optimal serverless model serving environment. There have been lots of work which advance the performance of FaaS execution environment. However, for ordinary users, it is very challenging to setup and try newly added features to improve the service throughput. AYCI tries to fill the gap by providing various pre-built environments which applying the state-of-the-art technologies in FaaS to serve DNN models. Using AYCI, users provide their custom DNN models which can be executed on various configurations. AYCI provides detailed performance metrics to users so that they can decide whether it is worth to take extra time for further optimization in a specific configuration in FaaS model serving.

Figure 1 shows the overall architecture of the proposed service. It is mainly composed of *FrontEnd, Data Storage, Runtime Arbitrator*, and *Output Reporter*. The *FrontEnd* module is responsible for providing a webpage to access the service. The module takes a DNN model to execute as an input argument from a user. In addition, users can specify FaaS run-time configurations, such as target hardware, model optimizer, hardware compiler, batch sizes, and memory sizes. AYCI stores prior execution results and performance metrics of a submitted model in a permanent storage for future reference. To identify a model, AYCI performs SHA-256 hashing for a submitted model and concatenates the model size to reduce the hash value collision possibility. Upon submission of a model from an user, the *FrontEnd* checks whether the model has already been executed in the *Data Storage* module.

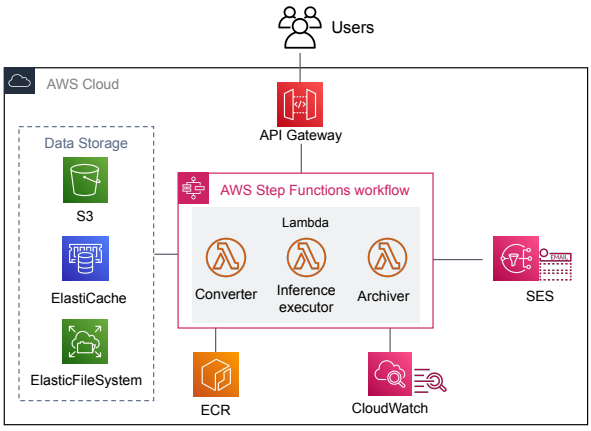
The *Data Storage* module is responsible for storing input, output, and intermediate files. To reuse prior execution records, a result archive stores all the performance metrics in a key-value format. User-submitted models are stored in a permanent file storage. When a model needs optimization, an intermediate compiled model is also stored in the module. The *Data Storage* module is also responsible for storing and serving container images as a FaaS run-time environment. The FaaS run-time should be prepared differently according to the underlying hardware types, DNN development platforms of different versions, model optimization techniques.

The *Runtime Arbitrator* module orchestrates the overall process of DNN model inference using FaaS. If a user selects a model optimization testing, the module selects the appropriate model conversion environment and triggers model compilations. In the model converter, the input is the user-submitted model from an user, and the output is the optimized model which is stored in the *Data Storage* module. A target model for execution, either user-submitted or optimized, is executed on FaaS with configurations specified by users. The metric collector in the *Runtime Arbitrator* module gathers performance metrics during the model inference execution. The collected metrics include the inference latency and throughput, model graph optimization latency, hardware-specific compilation time, model loading time, library loading time, and maximum memory usage. The DNN model prediction accuracy is dependent on the test dataset, and AYCI does not collect accuracy-related metrics.

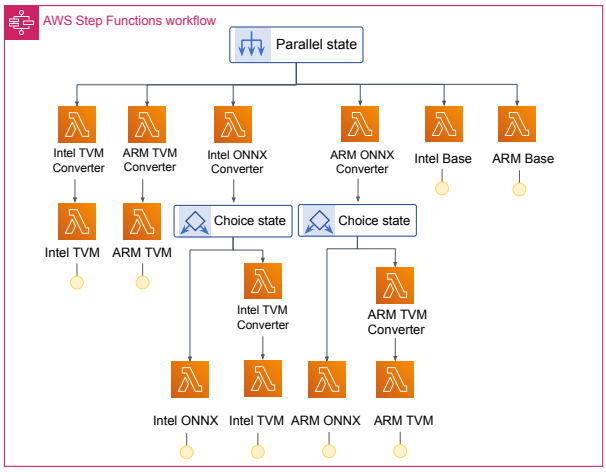
The *Output Reporter* module is responsible for presenting inference metrics to users. The model conversion and inference can take time, and the result is presented asynchronously. After gathering performance metrics, the module can send the summarized result to the submitter via an e-mail. The execution result is stored in the result archive where users can query the previous execution performance metrics.

## 4 SERVICE IMPLEMENTATION

We have implemented the prototype of All-You-Can-Inference using AWS which is shown in Figure 2, and the web service and source codes are publicly available - https://github.com/ddps-lab/lambda-optimize-serving. The prototype service is built adopting a serverless architecture. As shown in Figure 2a, a static frontend web-page is served from an object storage service, S3. Using the provided web interface, an user submits a DNN model for testing with various configurations. With the current prototype implementation with




**(a) All-You-Can-Inference implementation using AWS**




**(b) The composition of Lambda functions of *Runtime Arbitrator* module**

**Figure 2: The implementation of All-You-Can-Inference as a service**

AWS, users can select target hardware types (Intel and ARM), model optimizer and hardware compiler (ONNX [10] and TVM [8]), DNN model development platforms (PyTorch [23], TensorFlow [1], and MXNet [7]), inference data batch size, and configured memory size.

For input and intermediate model storage, AYCI currently supports an object storage service, S3. To use an object storage service as a result archive, AYCI uses the composition of hash of input model and the size as a key of an object where the value is the prior execution result. To run inference tasks using FaaS, necessary packages should be prepared in a storage. Among possible options, AYCI adopts embedding all the libraries in a FaaS run-time container image which is stored in Amazon Elastic Container Registry (ECR). Using the container image, Lambda run-time is invoked for different purposes of model optimization converter and executor. For model graph optimization, AYCI uses ONNX which performs graph-level transformations, small graph simplifications, node eliminations and layout optimizations [10]. For hardware specific model

compilation, AYCI uses TVM. For the Intel hardware run-time, it uses the TVM compilation option of hardware target named *core-avx2*. For the ARM hardware run-time, it uses the compilation option of *arm_cpu()* provided by TVM API. Using ONNX and TVM optimization is not exclusive, and users can select to optimize using both libraries. The inference execution Lambda performs serving tasks using a submitted or optimized model. With the performance metrics from inference executions, performance estimation Lambda runs multivariate polynomial regression. The performance metrics are collected through Amazon CloudWatch, and the final result is sent by an e-mail with the Simple E-Mail Service (SES).

Figure 2b presents an example implementation of *Runtime Arbitrator* module in detail. In the prototype implementation, AYCI uses AWS Step Functions which allows orchestration of multiple Lambda functions to construct applications with a complex logic in a Directed Acyclic Graph (DAG) format [3]. The model optimization and inference execution sequence of AYCI can be well represented using DAG. For example, if a user selects model optimization using ONNX followed by hardware-specific compilation using TVM with a vanilla model execution, the vanilla model execution and model optimization can happen in parallel. Two consecutive model optimization can be expressed sequentially. Figure 2b shows the Step Function implementation with PyTorch. Please note that different DNN development platforms should use distinct function execution container images, and the Step Function composition should be built separately. Based on the users' selection of hardware and optimization heuristics, the execution path is determined in a conditional manner. After executing inference task, the result and performance metrics are sent to a result archiving function for future reference. In the AWS Lambda configuration, changing the memory size might not be reflected in real-time and might cause unexpected cold-start [27], and AYCI deploys distinct Step Functions sets for pre-defined memory sizes.

## 5 INSIGHTS FROM SYSTEM EVALUATION

In order to evaluate the effectiveness of using AYCI in the serverless DNN model inference, we conducted experiments with various DNN models, *SqueezeNet, ShuffleNet, MobileNetV2, MNasNet, EfficientNetB0, ResNet18, ResNet50, InceptionV3, AlexNet, VGG16*, and *BERT*. To present performance variations while using different hardware, we use *Intel* and *ARM* CPUs that are available by AWS Lambda. Regarding the model optimizations, we use *ONNX* and *TVM*. To remove the impact from the cold-start and present consistent result, we excluded the result from the first execution and record next 11 execution result.

Figure 3 compares the performance of CNN and Natural Language Processing (NLP) models. We use *VGG16* and *BERT* models for CNN and NLP, respectively. The model size of both models is similar. The vertical axis shows the relative latency which is normalized to the best performing case in each configuration. We set the memory size of Lambda as 10GB where both models could execute successfully. We could uncover that both models show similar performance variations for both Intel and ARM CPUs. The Vanilla environment was the most effective mechanism with Intel hardware, while the TVM optimization was the most effective with ARM for both models. Observing the similar performance pattern of CNN and NLP, we decided to focus on evaluating the CNN performance for concise presentation.

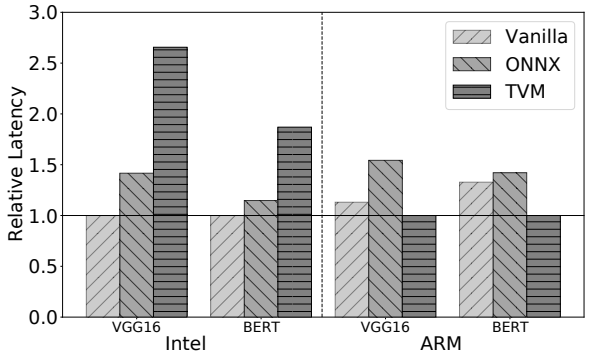



**Figure 3: Comparison of CNN and NLP performance variations in Intel and ARM serverless environments**

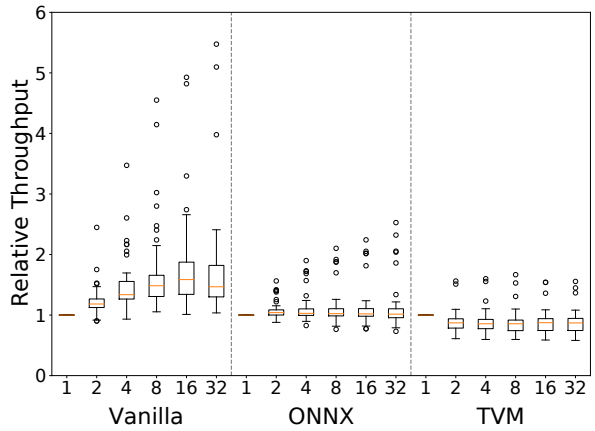



**Figure 4: Performance patterns when applying batch processing with serverless compiler configuration**

**Observation 1**: CNN and NLP models show similar performance patterns with different hardware and model optimization heuristics.

Batching multiple inputs for an inference task are expected to provide performance benefits in many cases [2], and we conduct experiments to see the impact of applying batching with diverse model optimization heuristics. Figure 4 shows how inference task performance changes as we enable batching with different size for diverse optimization heuristics which are shown in the horizontal axis. The vertical axis shows the relative throughput which is normalized to the throughput when the batch size is one. We show the throughput using a box-whisker plot with execution result of 10 CNN models, where each model is measured ten times, and we varied the memory size of Lambda as 0.5, 1, 2, 4, 8, and 10*GB*.

We can observe that the vanilla environment shows the most performance improvement with batching. When the batch size is 16, the throughput increased about 1.58×. The throughput of ONNX and TVM does not increase as much as the vanilla environment does. We can expect that the performance when applying ONNX

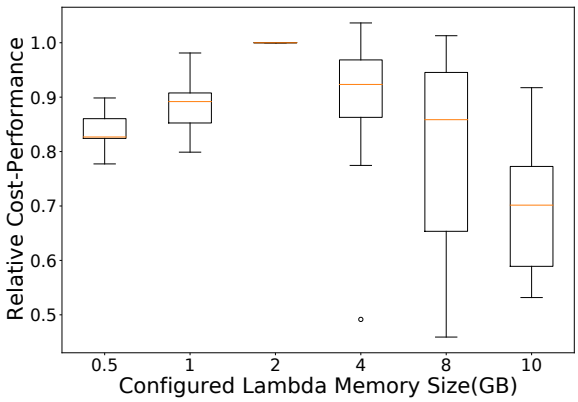



**Figure 5: Efficient memory allocation in serverless environment**

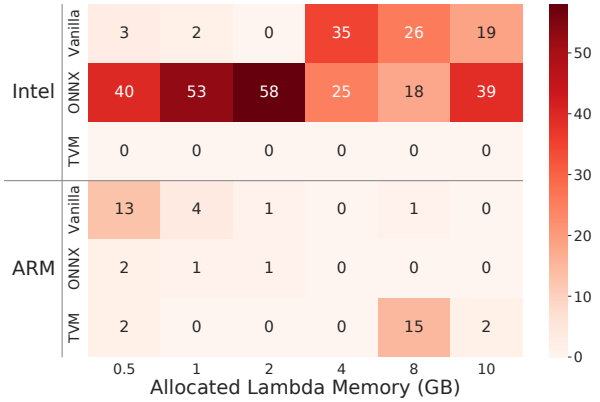



**Figure 6: The number of most efficient cost-performance cases of different configurations**

or TVM has already improved by the model optimization, and the benefit from a larger batch size is not as remarkable as the vanilla environment.

**Observation 2**: Batching DNN inference results in throughput improvement especially in a vanilla serverless environment, but the batching shows negligible performance gain when model optimization heuristics are applied.

When using Lambda, setting the proper memory size is very important to achieve good performance in a cost-efficient way. To present the performance variations with different Lambda memory size, Figure 5 present how the CNN models behave differently as the configured Lambda memory size changes. The horizontal axis shows the configured memory size. The vertical axis shows the relative cost-performance normalized for the value when memory size is 2GB for an arbitrary memory size, $M$GB, which is calculated as the $\frac{M}{2} \times \frac{Latency(M)}{Latency(2)}$. The intuition from the metric is as follow. The increase in the memory size results in the proportional cost increase, and whether the memory size increase can result in larger performance gain that the cost increase. We select the base case 2GB RAM because it shows the best cost-performance.

In most models, the best performance achieved when the configured RAM size is 2GB in a vanilla environment. As the configured memory size becomes larger, we can notice that the cost-performance metric drops quite significantly. We can conjecture that for most DNN model inference tasks, increasing the memory and CPU capacity does not result in proportional performance gain while incurring larger cost. In the experiment, we use various models with distinct sizes. Regardless of the model size, setting 2GB RAM can result in decent DNN inference performance in most cases.

**Observation 3**: In a serverless DNN inference, setting the memory size of Lambda to 2GB results in the best performance in most cases.

In order to present the best performing DNN inference environment, we run inference jobs in all possible cases of 10 CNN models. We counted the number of best performing cases for each hardware-optimization combination and show the result in Figure 6. In most cases, Intel-ONNX combination achieves the best performance, especially when the configured Lambda memory size is small. For larger Lambda memory sizes, we can observe that the ARM-TVM and Intel-Vanilla options often perform the best. From this experiment, we can conclude that ARM hardware is not as efficient as Intel hardware as the service provider announces. This observation contradicts with previous work which presents the superb performance of serverless ARM hardware for NLP data pipeline tasks [20]. In the previous work, the authors did not include various model optimization heuristics, such as ONNX, as we did. Due to the growing development eco-system of ARM hardware, the optimization library might be still in the infant stage. As the development eco-system matures, we expect superb cost-performance gain when using ARM hardware with an optimized model.

**Observation 4**: The DNN inference performance of ARM hardware is not as good as that of Intel hardware with an optimized model.

## 6 RELATED WORK

Serverless computing has advantages over traditional serverful environment especially when there are sudden high requests while serving deep learning models. In the context, MARK [30] proposed a hybrid system that utilizes on-demand spot instances to save cost and FaaS systems to handle resource interruption. BATCH [2] proposed a deep learning inference system based on a FaaS system. It focuses on proposing an optimal batching heuristic during serverless DNN inference. Comparing to the previous work, AYCI has much broader optimization scopes. Similar to AYCI, BentoML [11] automatically builds a container image which can be deployed on a public cloud environment. It adopts code-based service deployment approach, while AYCI adopts a fully-managed service approach with AWS step functions. Regarding using ARM CPU in FaaS, NLP pipeline workloads [20] showed that ARM hardware can reduce execution latency and achieve more reliable performance than on Intel hardware. The workload in the workload is rather simple and does not require heavy DNN inference packages which we expect to be the main reason for the performance difference.

FunctionBench [16] proposes various workloads on FaaS environment. Multiple optimization heuristics proposed in this work can be applied to FunctionBench for further performance improvement.

## 7 CONCLUSION AND FUTURE WORK

We propose All-You-Can-Inference, a fully-managed web-service which automates evaluation of DNN model inference on a diverse settings of serverless computing environment. With the presented system design and the prototype implementation, we showed the practicality of the proposed system uncovering challenges in the FaaS environment setup and performance variations for distinct models. We are certain that the proposed system can help users build an optimal serverless DNN inference system. The current system is in the early stage. From the implementation perspective, serving a model with with some specific environments, such as MXNet on an ARM FaaS, are not feasible yet, and we actively working to expand the supporting cases. The current implementation supports only AWS, and we are planning to support Microsoft Azure and Google Cloud. We could not gain noticeable performance gain when applying TVM for hardware-specific compilation because we have limited hardware information. We are working to improving the performance of applying TVM on FaaS.

## ACKNOWLEDGMENTS

This work was supported in part by the National Research Foundation (NRF) Grant funded by the Korean Government (MSIP) under Grants NRF-2022R1A5A7000765 and NRF-2020R1A2C1102544, and in part by the SW StarLab under Grant RS-2022-00144309 by IITP.

## 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] Ahsan Ali, Riccardo Pinciroli, Feng Yan, and Evgenia Smirni. 2020. BATCH: Machine Learning Inference Serving on Serverless Platforms with Adaptive Batching. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15. https://doi.org/10.1109/SC41405.2020.00073

[3] Ioana Baldini, Perry Cheng, Stephen J. Fink, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Philippe Suter, and Olivier Tardieu. 2017. The Serverless Trilemma: Function Composition for Serverless Computing. In *Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software* (Vancouver, BC, Canada) *(Onward! 2017)*. Association for Computing Machinery, New York, NY, USA, 89–103. https://doi.org/10.1145/3133850.3133855

[4] AWS Blog. [n. d.]. AWS Lambda – Container Image Support. https://aws.amazon.com/blogs/aws/new-for-aws-lambda-container-image-support/

[5] AWS Blog. last accessed April. 2020. AWS Lambda announces Provisioned Concurrency. https://aws.amazon.com/about-aws/whats-new/2019/12/aws-lambda-announces-provisioned-concurrency/

[6] Ermao Cai, Da-Cheng Juan, Dimitrios Stamoulis, and Diana Marculescu. 2017. Neuralpower: Predict and deploy energy-efficient convolutional neural networks. *Asian Conference on Machine Learning* (2017).

[7] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. *CoRR* abs/1512.01274 (2015). arXiv:1512.01274 http://arxiv.org/abs/1512.01274

[8] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 578–594. https://www.usenix.org/conference/osdi18/presentation/chen

[9] Google Cloud. [n. d.]. Cloud Functions Image Build. https://cloud.google.com/functions/docs/building

[10] ONNX Runtime developers. 2021. ONNX Runtime. https://onnxruntime.ai/.

[11] BentoML Documents. [n. d.]. What is BentoML? https://docs.bentoml.org/en/latest/

[12] Azure Functions. [n. d.]. Create a function on Linux using a custom container. https://docs.microsoft.com/en-us/azure/azure-functions/functions-create-function-linux-custom-image

[13] 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

[14] AWS What is new. [n. d.]. AWS Lambda support for Amazon Elastic File System now generally available. https://aws.amazon.com/about-aws/whats-new/2020/06/aws-lambda-support-for-amazon-elastic-file-system-now-generally-/

[15] Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. 2017. Serving deep learning models in a serverless platform. *CoRR* abs/1710.08460 (2017). arXiv:1710.08460 http://arxiv.org/abs/1710.08460

[16] 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

[17] 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.

[18] J. Kim and K. Lee. 2020. I/O Resource Isolation of Public Cloud Serverless Function Runtimes for Data-Intensive Applications. *Cluster Computing* (2020). https://doi.org/10.1007/s10586-020-03103-4

[19] Jeongchul Kim, Jungae Park, and Kyungyong Lee. 2019. Network Resource Isolation in Serverless Cloud Function Service. In *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W)*.

[20] Danielle Lambion, Robert Schmitz, Robert Cordingly, Navid Heydari, and Wes Lloyd. 2022. Characterizing X86 and ARM Serverless Performance Variation: A Natural Language Processing Case Study. In *Companion of the 2022 ACM/SPEC International Conference on Performance Engineering* (Bejing, China) *(ICPE '22)*. Association for Computing Machinery, New York, NY, USA, 69–75. https://doi.org/10.1145/3491204.3543506

[21] H. Lee, K. Satyam, and G. Fox. 2018. Evaluation of Production Serverless Computing Environments. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, Vol. 00. 442–450. https://doi.org/10.1109/CLOUD.2018.00062

[22] J. Park, , H. Kim, and K. Lee. 2020. Evaluating Concurrent Executions of Multiple Function-as-a-Service Runtimes with MicroVM. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*.

[23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[24] Hang Qi, Evan R. Sparks, and Ameet Talwalkar. 2017. Paleo: A Performance Model for Deep Neural Networks. In *Proceedings of the International Conference on Learning Representations*.

[25] 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.

[26] Mohammad Shahrad, Rodrigo Fonseca, Inigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 205–218. https://www.usenix.org/conference/atc20/presentation/shahrad

[27] 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

[28] Geoffrey X. Yu, Yubo Gao, Pavel Golikov, and Gennady Pekhimenko. 2021. Habitat: A Runtime-Based Computational Performance Predictor for Deep Neural Network Training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 503–521. https://www.usenix.org/conference/atc21/presentation/yu

[29] Tianyi Yu, Qingyuan Liu, Dong Du, Yubin Xia, Binyu Zang, Ziqian Lu, Pingchao Yang, Chenggang Qin, and Haibo Chen. 2020. Characterizing Serverless Platforms with Serverlessbench. In *Proceedings of the 11th ACM Symposium on Cloud Computing* (Virtual Event, USA) *(SoCC '20)*. Association for Computing Machinery, New York, NY, USA, 30–44. https://doi.org/10.1145/3419111.3421280

[30] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. 2019. MArk: Exploiting Cloud Services for Cost-Effective, SLO-Aware Machine Learning Inference Serving. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, Renton, WA, 1049–1062. https://www.usenix.org/conference/atc19/presentation/zhang-chengliang