EdgeServe: Efficient Deep Learning Model Caching at the Edge

Tian Guo
Worcester Polytechnic Institute
Worcester, Massachusetts
tian@wpi.edu

Robert J. Walls
Worcester Polytechnic Institute
Worcester, Massachusetts
rjwalls@wpi.edu

Samuel S. Ogden
Worcester Polytechnic Institute
Worcester, Massachusetts
ssogden@wpi.edu

ABSTRACT

In this work, we look at how to effectively manage and utilize deep learning models at each edge location, to provide performance guarantees to inference requests. We identify challenges to use these deep learning models at resource-constrained edge locations, and propose to adapt existing cache algorithms to effectively manage these deep learning models.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Computer systems organization** → **Cloud computing**; • **General and reference** → **Performance**.

KEYWORDS

Deep Learning Inference, Caching Algorithm, Edge Computing, Performance Optimization

ACM Reference Format:
Tian Guo, Robert J. Walls, and Samuel S. Ogden. 2019. EdgeServe: Efficient Deep Learning Model Caching at the Edge. In *SEC '19: ACM/IEEE Symposium on Edge Computing, November 7–9, 2019, Arlington, VA, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3318216.3363370

1 INTRODUCTION

Motivation. Deep neural networks (DNNs) are increasingly leveraged to provide rich features—such as real-time language translation, image recognition and personal assistants [4, 18, 20]—in end-user applications. Furthermore, developers of these rich models often eschew the traditional execution model of performing inference on the end-user device and, instead, deploy these models on remote servers [7, 8, 16]. This switch to the cloud confers a number of benefits including the ability to benefit from complex models that are infeasible to execute on relatively resource-constrained end-user devices [1, 11, 14]; and the ability to maintain the confidentiality of models trained on proprietary datasets [12, 19].

The benefits of cloud-based deep learning come at the cost of novel challenges. For example, in order to use cloud-based deep neural networks, applications must transmit and receive inference requests and responses over highly variable networks; Figure 1 shows that the time to send a small image file can vary from 148ms to 1405ms in LTE. Such variation is magnified when sending larger inference requests, e.g., audio or video clips, leading to unpredictable end-to-end response times and making it challenging to meet quality-of-service requirements.

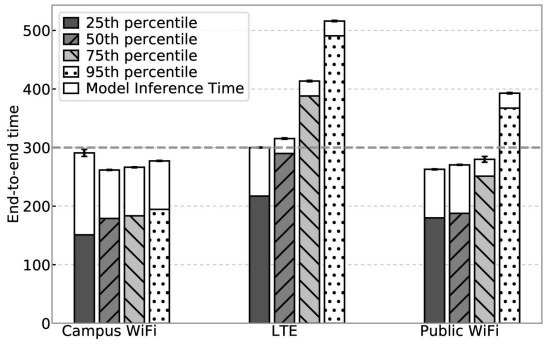

Figure 1: *Network variations and benefits of functionally-equivalent DNN models.* *We measured the time for a Pixel2 running Android Oreo to send a 330KB image. We repeated the measurement over 700 times for each network type.*

In this work, we position ourselves as a hypothetical CDN provider that wants to provide a novel feature for their clients: *model execution caching*. Intuitively, model execution caching refers to a class of solutions that execute the deep learning models on small server clusters geographically close to end-users (i.e., edge clouds), thereby reducing the impact of network delays and variance. We adopt the term caching, as the solutions in this space must make resource decisions that are reminiscent of traditional caching problems. For example, one challenge is deciding which models should be loaded into the memory of the limited number of available servers to maximize the probability of servicing an incoming request without swapping models in and out of memory, i.e., maximizing the chance of a *cache hit*.

For the ease of exposition, we frame our discussion in the context of a hypothetical system for model execution caching called *EdgeServe*. The design challenges of EdgeServe cover multiple aspects. First, EdgeServe needs to manage a large number of deep learning models. The number of these models scale with the number of applications, as well as the number of *functionally-equivalent* models. Here, we define two models as functionally-equivalent if they can be used to service the same type of requests, e.g., image classification. As we found in previous work [15] and show in Figure 1, applications often benefit from having access to a set of functionally-equivalent deep neural networks that differ in execution time. Second, edge clouds are inherently resource constrained—having fewer servers than traditional clouds. Even though it might be possible to store all DNN models at each edge cloud, it is often not feasible to keep all DNN models in the main and GPU memory. Third, the cost of a *model miss*—dominated by the time it takes to

SEC '19, November 7–9, 2019, Arlington, VA, USA

ACM ISBN 978-1-4503-6733-2/19/11.
https://doi.org/10.1145/3318216.3363370

	accuracy (%)	inference time model hit (ms)	inference time model miss(ms)	size (MB)	#params (M)
SqueezeNet	72.9	28.6 ± 1.1	173.4 ± 25.7	4.8	1.2
MobileNetV1 0.25	74.1	25.7 ± 1.2	272.8 ± 45.0	1.9	0.5
MobileNetV1 0.5	84.9	26.3 ± 1.2	302.8 ± 45.5	5.2	1.3
DenseNet	85.6	49.6 ± 3.2	1149.0 ± 108.0	43.9	-
MobileNetV1 0.75	88.1	28.0 ± 1.1	351.9 ± 47.4	10.5	2.6
MobileNetV1 1.0	90.6	28.2 ± 1.2	421.2 ± 47.1	17.1	4.2
NasNet Mobile	91.5	55.3 ± 4.1	2817.2 ± 123.7	21.9	5.3
InceptionResNetV2	94.0	76.3 ± 5.7	2844.3 ± 106.5	121.6	55.8
InceptionV3	93.8	55.8 ± 1.2	1950.7 ± 101.2	95.7	23.8
InceptionV4	95.1	82.8 ± 0.9	3162.2 ± 134.0	171.2	42.7
NasNet Large	96.1	112.6 ± 6.1	7054.5 ± 238.4	356.6	42.3

Table 1: *Statistics of popular CNN Models.* *We measured the average inference time using an EC2* `p2.xlarge` *GPU server with 12GB GPU memory.*

load models from storage to memory—can vary from hundreds of milliseconds to a number of seconds, as shown in Table 1.

The reader might wonder why we advocate caching deep learning models instead of caching the inference results. The latter is both more reminiscent of traditional CDN functions and attractive from a performance perspective as it suggests EdgeServe could service requests without needing to re-execute the model. However, we focus on model caching as it is better suited to workloads that are often unique and user-specific [17], e.g., images or audio clips. Directly caching these inference results for unique user-specific requests would not be very useful as there is a high likelihood that EdgeServe will not be able to reuse these results, leading to few cache hits and frequent misses. While it might be possible to improve hit rates by designing sophisticated schemes to perform non-trivial preprocessing [9] of incoming requests in order to map to previously cached results [21], we argue that model execution poses fewer demand on the model developer and thus enjoys a lower barrier to adoption.

2 CACHING DEEP LEARNING MODELS AT THE EDGE

Background. To leverage deep learning models hosted on edge servers, end-user applications need to send inference requests, together with input data, that specify which deep learning model is required to service the request. These requests are then processed by the edge cloud. We assume that requests are first processed by a dispatching server [2] that sits in front of a deep learning framework, such as TensorFlow [16] or Caffe [3]. Depending on the underlying server capacity, e.g., memory and GPU, these frameworks often have access to multiple in-memory DNN models. If the requested model currently resides in memory on some server in the cluster, the framework can start executing the inference request immediately. Otherwise, the framework needs to first load the specified model and then generate the inference result. We refer to the former case as a *model hit* and the latter as a *model miss*.

As loading models into memory takes non-trivial time (see Table 1), an alternative is to forward the request to an edge location that has the model already loaded. This scenario suggests the utility of having either centralized or distributed coordination among different edge locations. Regardless, when a model miss happens, the end-to-end design goals of EdgeServe is to create a suite of model cache algorithms that reduces model miss frequency and cost.

Problem Statement. EdgeServe will cache models in edge server memory with the goal of providing consistent response times for inference requests from end-user applications. We envision that EdgeServe will leverage existing edge clusters—potentially as part of existing CDN infrastructure—and aim to provide the model caching as a service for customers (i.e., developers). Caching deep learning models, at its core, is to move computation closer to the end-users. EdgeServe must efficiently decide which DNN models to preload into the memory and which to evict when the memory is full, all while meeting end-to-end inference response time requirements. Here efficiency means that EdgeServe's model management algorithms should be lightweight *relative* to the network transfer time and model execution time, both of which can take hundreds of milliseconds. Compared to traditional caching problems, like virtual address translation in operating systems or in-memory object caches like memcached [13], EdgeServe can afford more complex caching eviction and replacement algorithms. We assume that we have a known, but limited, number of servers at each edge cloud location; however, dynamically changing the edge cluster size (i.e., the cache size) represents an interesting avenue for future work.

Potential Solutions. Given that at the core of EdgeServe is a caching problem, we propose to first evaluate the effectiveness of existing cache replacement algorithms [6, 10, 22] in managing deep learning models at edge locations. To leverage prior cache algorithms, we need to adapt them to the domain of DNN models by defining the utility and cost of a cache miss.

Before the edge server's memory capacity is reached, we propose to leverage historical model usage patterns to pre-load popular DNN models into memory. Here, DNN model usage statistics can be recorded directly by EdgeServe. To decide which models to evict from memory, we need to have access to runtime model memory utilization. These could be obtained by instrumenting the deep learning frameworks [5] or estimating through stress test offline. Moreover, we need to quantify the utility of DNN models and the cost of *model misses*. We propose to use DNN model accuracies and computation complexities to derive the utility, e.g., highly accurate and faster DNN model will have higher utility. We further propose to use DNN runtime memory consumption and the time to load DNN models into memory as two key aspects of *model misses* cost.

Preliminary results. In this work, we conducted an initial measurement study to demonstrate the feasibility and illustrate the challenges of caching models at edge clouds. We focus on a popular type of model, convolutional neural networks (CNNs), that are often used for image classification. We looked at more than ten popular pre-trained CNN models and reported their inference accuracies and model sizes in Table 1. We summarize our initial observations below.

First, we observe that the persistent storage requirements are relatively modest as it is possible to store tens of thousands of CNN models at each edge location. Importantly, this implies that each edge location might be able to store the complete set of models and thus we need not consider (at least initially) the problem of distributing models amongst the clusters.

Second, the *model miss* overhead not only prolongs the execution time by up to 63X, but also leads to more time variation—ranging from a few milliseconds to hundreds of milliseconds—making it

harder to predict the impact of a *model miss*. To obtain these results, we set up the smallest GPU server in Amazon EC2 to host the models. To quantify the overhead of a *model miss*, we measured the execution time with *model hit* and *model miss* of using each CNN model over 3000 input images. Note, the difference between model hit and miss is whether the CNN model has already been loaded into the server memory.

3 SUMMARY

In this work, we proposed the caching of DNN models at edge locations to improve performance without degrading inference accuracy. We empirically demonstrated the benefits of hosting DNN models in edge clouds and illustrated the key challenges in designing model caching systems.

Acknowledgments. We thank Guin R. Gilman for her help in proofreading the final version of this paper. This work is supported in part by National Science Foundation grants #1755659 and #1815619 and Google Cloud Platform Research credits.

REFERENCES

[1] List of hosted models. https://www.tensorflow.org/lite/models.

[2] Nvidia/tensorrt-inference-server: The tensorrt inference server provides a cloud inferencing solution optimized for nvidia gpus.

[3] Delivering real-time AI in the palm of your hand. https://code.facebook.com/posts/196146247499076/delivering-real-time-ai-in-the-palm-of-your-hand/, 2016.

[4] Deep Learning for Siri's Voice: On-device Deep Mixture Density Networks for Hybrid Unit Selection Synthesis. https://machinelearning.apple.com/2017/08/06/siri-voices.html, 2017.

[5] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

[6] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter. Adaptsize: Orchestrating the hot object memory cache in a content delivery network. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*.

[7] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv:1512.01274*.

[8] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica. Clipper: A low-latency online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI'17)*.

[9] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805*.

[10] B. S. Gill and D. S. Modha. Sarc: Sequential prefetching in adaptive replacement cache. In *USENIX Annual Technical Conference*, 2005.

[11] T. Guo. Cloud-based or on-device: An empirical study of mobile deep inference. In *2018 IEEE International Conference on Cloud Engineering (IC2E '18)*.

[12] F. Mo, A. S. Shamsabadi, K. Katevas, A. Cavallaro, and H. Haddadi. Towards characterizing and limiting information exposure in DNN layers. *arXiv:1907.06034*.

[13] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani. Scaling memcache at facebook. In *the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*.

[14] S. S. Ogden and T. Guo. MODI: Mobile deep inference made efficient by edge computing. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*.

[15] S. S. Ogden and T. Guo. Modipick: Sla-aware accuracy optimization for cloud-based inference. *arXiv:1909.02053*, 2019.

[16] C. Olston, N. Fiedel, K. Gorovoy, J. Harmsen, L. Lao, F. Li, V. Rajashekhar, S. Ramesh, and J. Soyke. Tensorflow-serving: Flexible, high-performance ML serving, 2017.

[17] S. Ramaswamy, R. Mathews, K. Rao, and F. Beaufays. Federated Learning for Emoji Prediction in a Mobile Keyboard. *arXiv:1906.04329*.

[18] A. van den Oord, S. Dieleman, and B. Schrauwen. Deep content-based music recommendation. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2643–2651. Curran Associates, Inc., 2013.

[19] P. M. VanNostrand, I. Kyriazis, M. Cheng, T. Guo, and R. J. Walls. Confidential Deep Learning: Executing Proprietary Models on Untrusted Devices. *arXiv:1908.10730*, 2019.

[20] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv:1609.08144*.

[21] M. Xu, M. Zhu, Y. Liu, F. X. Lin, and X. Liu. Deepcache: Principled cache for mobile deep vision. In *the 24th Annual International Conference on Mobile Computing and Networking (MobiCom '18)*.

[22] J. Yang, R. Karimi, T. Sæmundsson, A. Wildani, and Y. Vigfusson. Mithril: Mining sporadic associations for cache prefetching. In *the 2017 Symposium on Cloud Computing (SoCC '17)*.