Couper: DNN Model Slicing for Visual Analytics Containers at the Edge

Ke-Jou Hsu
Georgia Institute of Technology
nosus_hsu@gatech.edu

Ketan Bhardwaj
Georgia Institute of Technology
ketanbj@gatech.edu

Ada Gavrilovska
Georgia Institute of Technology
ada@cc.gatech.edu

ABSTRACT

Applications incorporating DNN-based visual analytics are growing in demand. This class of data-intensive and latency-sensitive workloads has an opportunity to benefit from the emerging edge computing tier. However, to decouple the growing resource demand of DNN models, from the characteristics and resource limitations of the infrastructure elements available at the edge, new methods are needed to quickly slice the DNNs into appropriately-sized components, and to deploy those DNN slices to be executed on the edge infrastructure stacks. This paper presents Couper, a practical solution that provides for quick creation of slices of production DNNs for visual analytics, and enables their deployment in contemporary container-based edge software stacks. Couper is evaluated with 7 production DNNs, under varying edge configurations. The result demonstrates that, when compared to traditional cloud systems, Couper can provide up to 90% improvement in processing latency, and nearly 100% improvement in processing quality, while reducing the search space for a solution by 99%.

ACM Reference Format:
Ke-Jou Hsu, Ketan Bhardwaj, and Ada Gavrilovska. 2019. Couper: DNN Model Slicing for Visual Analytics Containers at the Edge. In *Proceedings of SEC 2019: The Fourth ACM/IEEE Symposium on Edge Computing (SEC '19).* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

remember to replace URL in the ACM reference to the URL you would have received when I think you did the copyright forms

1 INTRODUCTION

Video formed over 80% of all Internet traffic in 2018. By 2021, ~250 exabytes data traversing the Internet will majorly comprise of visual data [22]. Real-time analysis of that visual data is a high value target as it is central to enabling several classes of important applications, including surveillance and security, traffic monitoring and prediction, shopping assistants, to providing care for seniors, infants or even pets.

Key technology enabling these use cases are deep neural networks (DNNs) that have replaced classical computer vision methods due to their accuracy and effectiveness [34, 38, 56, 57]. Unfortunately, using DNNs to analyze visual data is prohibitively expensive at scale. The scale manifests itself in three ways: First, the decrease of image sensor costs increases the amount of available visual data to be analyzed. To analyze all the visual data using DNNs deployed in the cloud, data has to be transported from cameras to centralized cloud datacenters, posing bandwidth demands on the backhaul network. Second, cloud latencies are not sufficient for the (near) real-time nature of some of the driving applications [15, 20, 33, 44]. Both the bandwidth demand and the latency requirements challenge the ability of cloud-based deployments to support these services. Third, the DNNs used for analysis of visual data are continuing to get deeper and more complicated (e.g., Microsoft's ResNet (2015) has a few hundreds of layers vs. Google's NASNet (2017) has more than one thousand layers for increased accuracy). The dominant cost in production DNN-based visual data analysis is serving predictions from increasingly more complex models. Expecting that they can always be deployed on end devices alone is not practical. In summary, there exists a wide gap among the demand that real-time analysis of visual data puts on infrastructure capabilities of current device–cloud deployment models, and the available compute and network resources. Deploying DNNs on shared and distributed edge computing infrastructure is a natural step in bridging this gap.

Visual analytics at the edge. Cloud service providers [16, 17, 23] and mobile network operators [36, 47, 52] are providing or developing shared infrastructure for deploying such applications at the edge of the network [30]. Therefore, in theory, the above mentioned latency- and bandwidth-related problems will not impede real-time visual analysis applications. However, expecting that the resources available as part of the edge infrastructure will always be sufficient for a full DNN model, is not realistic. First, the hardware technologies considered for the edge tier vary widely in their resource capabilities [27, 51]. Second, the edge may be shared by multiple tenants [30]. A resource-limited and multi-tenant edge cannot be relied upon to always provide the resources needed for arbitrary production models, when at the same time the demand for DNN models, and their number and complexity will continue to increase [55].

Gaps in current solutions. To address these limitations, ongoing research has taken two main approaches. One approach is to "fit" DNN models and their execution within the available edge infrastructure capabilities. These include model-level optimizations and parameter tuning to achieve a desired accuracy and resource footprint tradeoff [66] or to maximize the number of models supported from an edge location [37]. These solutions aim to create and deploy a new model, from potentially a very large search space, that will best fit given edge resources. The time and effort required for

SEC '19, November 07–09, 2019, Washington DC

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

model tuning limits the practical use of these approaches. Instead, applications will use pre-trained or production models, and the question is how to make it possible for such models to still benefit from an arbitrary edge. Even as new accelerators specialized for DNNs augment the edge nodes' capabilities [50, 53], the AI community is on an exponential curve of its own [46], and new models will continue to emerge with different resource requirements.

Model partitioning forms the basis of the second type of approaches. Several such efforts have developed methodologies to determine how to split a given DNN into components which optimize a particular metric, such as inference time, data movement latency, or energy-efficiency [39, 41]. However, searching for an ideal split point may not be practical, given the variability of the network characteristics, and fluctuations in the demand and availability of edge resource due to multi-tenancy. Therefore, one important question is how to quickly determine sufficiently good partitions. More importantly, however, these prior works do not tackle the question of how those model partitions can be dynamically configured and deployed as a pipeline of operations that will be applied to data being streamed across edge and cloud components.

Summarizing, an open problem is, *given a model and a set of edge resources, how to quickly determine the best way to partition the model* **and** *create deployment-ready model partitions needed to seamlessly run the model pipeline across the edge and the cloud?*

In response, we develop Couper – a practical solution for rapidly partitioning visual analytics DNN models for a given edge-cloud configuration. Couper contributes novel container-based infrastructure for model partitioning and deployment, applicable to most edge software stacks that use containers to deploy applications at edge infrastructure elements. Given a model and an edge configuration, Couper first determines and evaluates a small number of sufficiently good split point candidates, in order to quickly select among them the best one for a target metric. Next, Couper packages the model slices into deployment-ready containers for edge software platforms [13, 14, 19], augmented with a data relaying functionality needed to (de-)serialize and (re)direct data flows across the pipeline components.

Couper is built on the following key principles. First, it adopts *pipelining based on model slicing* as a core mechanism to decouple the requirements of a DNN-based visual analytics application, from the specific capabilities of the infrastructure elements where it can be deployed. Second, it adopts *separation of mechanisms from policies*, by incorporating mechanisms for model slicing and guided model evaluation, but specific algorithms to be used for slicing, evaluation and decision, are specified as inputs in Couper. We provide insights into important properties of such algorithms that reduce the number of split points which Couper must evaluate before reaching a decision. One insight is to take into consideration the properties of the neural network model, and to support *slicing algorithms focused on articulation points*. Slicing at these points avoids synchronization-related complexities; synchronization is shown to be expensive in geo-distributed settings [35], such as across edge nodes and the cloud, and should be avoided. A second insight is to use *edge-aware slice selection methods* which avoid evaluating split points known not to lead to benefits from running on an edge. Finally, Couper aims to provide *deployment readiness*: Solving the partitioning problem quickly is one part of the gaps in current systems. Making sure that those partitions can be deployed and executed in device-edge-cloud systems must also be solved.

Prior research has shown that model slicing is a practical path forward in leveraging the edge for DNN based visual analytics, since it decouples the resource limitations and characteristics of the edge infrastructure, from its ability to run increasingly deeper DNNs (Section § 3). However, designing a practical model slicing solution must consider the intricacies of the DNN structure, the streaming application frameworks and the deployment technology, which makes it extremely difficult. This is evident from the fact that a practical solution for slicing based on and for the edge is not available, despite the popularity of DNN based workloads and their use at edge. Couper demonstrates that it is possible to partition DNN-workloads at the system-level, independent of their structure, the deployment technology (containers) or application frameworks used to implement them, paving the way for independent exploration of system technologies focused on end-to-end considerations. In summary, this paper makes the following contributions:

- Couper is the first work, to the best of our knowledge, that describes general system-level support for DNN model slicing. In designing Couper, we took a systematic approach towards understanding the opportunity (Section § 3) to come up with a concrete design (Section § 4) focused on visual analytics models developed using TensorFlow [10] and the SAF streaming framework [65], suited best for deployments of container-based workloads orchestrated using Kubernetes [43].
- Second, Couper presents a functional framework that can be used to explore and evaluate different model slicing methods. We use it to develop a number of such methods (Section § 5) that reduce the complexity of making a good slicing decisions by orders of magnitude compared to a *strongman* approach, while resulting in model partitions that provide similar performance.
- Finally, using several popular and state-of-the-art production-ready DNN models, we demonstrate the utility of Couper. Couper requires minimal resources (Section § 6), and is effective in quickly determining split points for DNN model deployments that result in application performance gains through better use of the available edge computing infrastructure (Section § 7).

2 EDGE COMPUTING AND VISUAL ANALYTICS: A NATURAL FIT

The demand for new data-intensive services with low and predictable latencies are driving the emergence of a new edge infrastructure tier [27, 51]. There are certain characteristic of edge computing that motivated us to develop Couper in its current form.

The edge comes in many shapes and sizes. Edge computing is still in its nascent form. A number of point solutions for edge computing already exist [8], and industry is mobilizing with development of new enabling technology [4, 6] and with outlining the requirements of general purpose edge computing [30]. Many types of very diverse technologies are being positioned as possible edge platforms, with capabilities ranging from low-power processors [21, 67], densely positioned on rooftops and lamp posts, to custom racks for server-grade processor blades specialized for deployment in cellular base stations [12]. Depending on the location of deployment within the access network, at gateways, cellular base

stations, and other end points in the network data paths between devices and remote cloud datacenters, the edge resources and their network distance from the client devices and the cloud vary. As a consequence, *edge computing provides many trade-offs that have not been fully explored.*

The edge is a dynamic, shared resource. The edge infrastructure is envisioned to be (almost certainly) shared by services packaged in containers [13, 14, 19], running potentially different applications, or serving different tenants of a single type of service. Such shared use will further increase the effective *dynamism in the resource footprint available to a tenant for the execution of its edge workload.* As a result, in this paper we consider a range of configurations of edge computing platforms and different network distances among the client, edge and cloud, running containerized workloads.

DNN based visual analytics: The killer app for the edge. Visual analytics is considered the killer app for edge computing due to the pressure it puts on the backhaul in terms of bandwidth, and due to its latency sensitivity, which can be impossible to address just with a remote cloud. *The growth in the complexity, depth and the number of deep neural network models being used for visual analytics is also pushing their compute requirements, and makes it difficult to reason about the performance of the models with different kinds of edge infrastructure.*

A different perspective on edge for visual analytics. Others have already argued for the importance of using the edge for visual analytics with low latency and low data movement costs [49, 54]. Instead, we ask the question, *how can visual analytics applications leverage the available edge to achieve desired performance goals.* Note that the desired performance goals may differ based on application-specific or operating constraints. They could be purely based on inference latency (i.e., the time to process and classify a visual image frame), or may be concerned solely with data movement costs, or the overall frame processing rate, or other factors such as energy usage, etc. For example, analysis of data from secure cameras deployed at public places to quickly spot unattended items (e.g., at airports) [68], or to detect suspicious vehicles in motion [32, 37], are scenarios where processing speed and latency are crucial. Other application such as the ones using drone-attached cameras for land surveys [2, 64] are primarily concerned with minimizing data movement cost and reducing bandwidth demand in analytics.

3 UNDERSTANDING THE OPPORTUNITY

To better understand the design space for Couper, we present our observations from evaluating an off-the-shelf visual analytics application under different deployment scenarios. This allows us to illustrate the tradeoffs concerning executing visual analytics models at the edge, and the requirements that need to be addressed with Couper's design.

Experimental testbed. We picked an off-the-shelf streaming visual analytics application developed using the *Streaming Analytics Framework*, SAF [65], and evaluated it with several production-ready DNN models. SAF allows creation of arbitrary pipelined visual analytics applications where users can customize their own application by stacking up operators. For instance, a basic image classification application in SAF has four stages in sequence: (1) *camera*, to extract a raw image from either device or file, (2) *transformer*, to transform a raw image into the input format of an ML model, (3) *DNN evaluator*, to run the model inference, and (4) *classifier*, to map the inference result to a label.

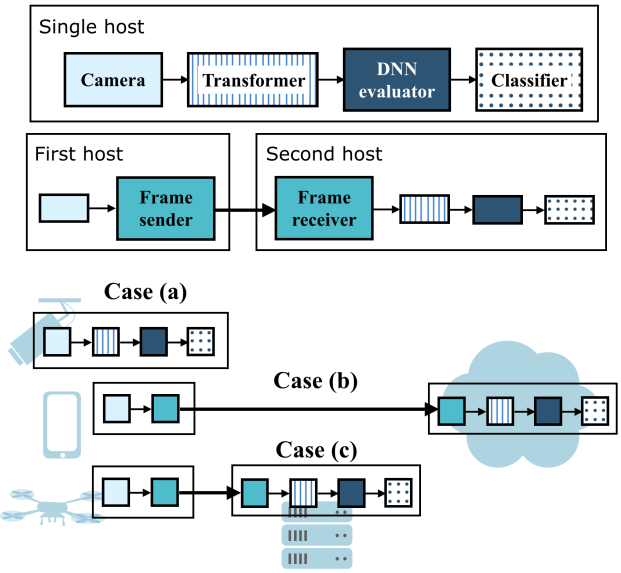


Figure 1: Image classification pipeline structure and three deployment cases: (a) running a full application including DNN inference on device only, (b) running the DNN inference on cloud, (c) running the DNN inference on edge.

Machines	CPU freq (GHz)	#CPU processors	RAM (GB)	GPU
Client	3.0	2	1	N/A
Edge	3.5	8	32	
Cloud	3.1	48	96	Nvidia P100

Table 1: Hardware specifications of the testbed

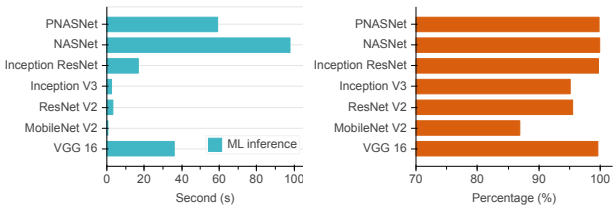
Parameters	Round-trip time (ms)	Jitter (ms)	Throughput (Mbps)
Client-to-edge	7.68	2.49	89.90
Edge-to-cloud	42.46	11.58	50.22
Client-to-cloud	49.20	13.39	41.02

Table 2: Network characteristics in the testbed

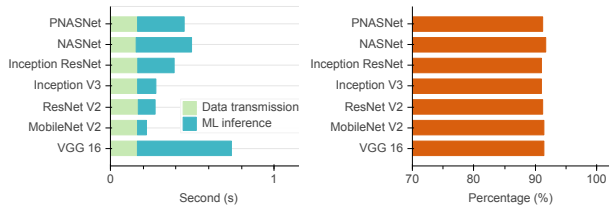
Figure 1 shows the different scenarios we evaluated: (a) run every stage on the client device to save communication cost, (b) offload the DNN processing to the cloud, to reduce inference time, and (c) offload the DNN processing to the edge. Figure 1 also illustrates the deployment of of the different stages of the image classification pipeline for each of the three configurations. While *camera* has to always remain on the device, *transformer*, *DNN evaluator* and *classifier* are run all on cloud (b) or all on edge (c). To connect the pipeline stages deployed across machines, SAF uses messaging operators (SAF provides ZMQ [24] and gRPC [62]) to send and receive frames.

Model	VGG 16	MobileNet V2 1.4	ResNet V2 50	Inception V3	Inception ResNet V2	NASNet 331	PNASNet 331
Top-1 Accuracy	71.5	74.9	75.6	78.0	80.4	82.7	82.9
# Operators	54	158	205	788	871	1265	939

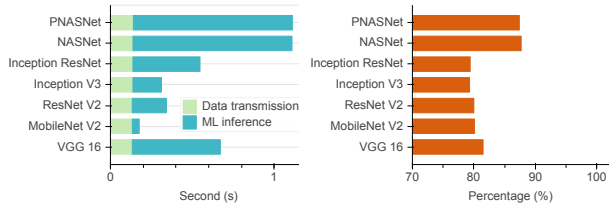
Table 3: The models with their accuracy and number of operators.



(a) Running ML inference on client



(b) Running ML inference on cloud



(c) Running ML inference on edge

Figure 2: Observed performance of the scenarios shown in Figure 1 using different DNN models. The green bars in left figures show end-to-end processing latencies; the orange bars on the right side correspond to frame drop rates.

The physical testbed is based on three server-class machines representing a client device, an edge node and a cloud server (Table 1). We orchestrate a container cluster across these machines, and deploy the application as a pipeline of Docker containers. We limit the resources used at the client and the edge by manipulating the CPU frequency and bounding the resources allocated to the containers running the client or edge functionality. In the more complete evaluation presented in Section § 7, we consider four different configurations for the edge nodes, based on possible deployment scenarios. We make an assumption that the client device and the edge machine are "nearby" (i.e., on the same LAN in our testbed), while the cloud server is remote (i.e., requires WAN), consistent with characterizations for edge computing presented elsewhere [45]. Table 2 summarizes the communication links between the nodes.

Using this testbed, we explore the following questions:

What are the edge benefits for visual inference? We use several popular DNN models (Table 3) with different complexity and accuracy to evaluate image classification in each of the above scenarios. The DNN models listed here are public TensorFlow pre-trained models [11]. Table 3 shows the number of operators in each model. An operator is the minimal computing unit of a model; within each model, operators are further grouped in layers corresponding to a specific type of DNN functionality, such as a layer combining variable square and summation operators [7]. From Table 3, we see that the models with higher accuracy tend to have more operators, and theoretically consume more computing resource for model inferencing.

Figure 2 shows the average latency of processing a frame based on repeatedly streaming the same video (see the left-hand-side figures with green bars). The latencies of the two most time consuming stages in the pipeline, model inference and data transmission, are shown stacked for the cloud and edge cases (Figure 2b and Figure 2c). The right-hand-side figures with orange bars show the frame drop rate. Note that we allow frames to be dropped in the pipeline to avoid backpressure between stages, since the camera processor continuously produces about 24 frames per second. An imbalanced pipeline drops more frames and affect the overall image processing quality [37]; a high frame processing rate is a desirable property for visual analytics applications.

We make the following key findings. First, models have different compute resource requirements to achieve the same SLA. Newer, more sophisticated models will continue to evolve and to deliver improved experiences for users; not one single type of machine can be guaranteed to suit every model. *Use of edge resources is critical for supporting models with resource requirements which exceed what is available on the client device alone.*

Second, although a high-end cloud server may result in shortest processing latency on deeper DNNs (from ResNet to PNASNet), the edge configuration is the one that is able to process more frames. Considering Figure 2a, in the client case there is no data transmission involved. However, frames are still dropped because of the lower processing speed of the device; i.e., the frame processing rate degrades for models with longer inference latency. In contrast, the cloud case (Figure 2b) experiences constant drop rates regardless of the model. This is because the bottleneck of the pipeline changes from compute to network I/O. Since *camera* keeps generating the same stream of raw image frames for each of the models, there is less divergence between models, and all experience similar amount of dropped frames. The edge case (Figure 2c) has more compute capacity than the client, and is able to reduce the frame dropping due to processing speed. The edge is connected to the camera via a faster network, thus reducing the number of network-related dropped frames, compared to the cloud. In summary, *the use of edge infrastructure provides opportunities to maintain higher frame processing rates, resulting in fewer dropped frames, and, consequently, higher application quality.*

Finally, for simpler models such as VGG and MobileNet, the edge node has sufficient resources to perform the inference just as fast as the cloud, which combined with the faster network, *results in both lower processing times and lower drop rates.*

How would model slicing make further improvements? Although the edge machine is closer to the camera and reduces data movement cost, it still degrades ML inference time on some models when compared to the cloud. For the edge configuration used in this experiment (Table 1), Inception, ResNet, NASNet and PNASNet exhibit 2× or higher slowdown, compared with running in the remote cloud with powerful servers with GPUs. Such slowdowns may not offset the value provided from the reduction in drop rate. Slicing the model into partitions which are then deployed as a pipeline across the edge and the cloud presents opportunities to achieve a tradeoff of reducing the frame drop rate, while still getting the benefits from the cloud's compute resources needed to maintain low processing times. The feasibility of achieving strictly improved processing time compared to a purely cloud-based deployment will depend on the properties of the communication links and on the resources available at the edge.

When considering slicing of a DNN across the edge and the cloud, the application pipeline is extended by breaking up the DNN evaluator block from Figure 1. This provides more flexibility in how the resulting pipeline stages are placed across the edge-cloud resources. Slicing not only requires another DNN evaluator but also additional pair of messaging operators to pass frames among the application components deployed on different machines. Figure 3 shows how an image classification application pipeline changes when the model is sliced: two pairs of message passing operators and an additional DNN evaluator are needed for a deployment involving three nodes, a client, edge and cloud.

To understand the tradeoffs concerning DNN slicing, for concreteness, we evaluate it with the ResNet v2 50 model, which has a large number of relatively simple operators. Among them, we considered 34 operators for possible split points in the model, based on our intuition. Figure 4 presents the end-to-end latency and frame drop rate observed with each split point. Each point on the x-axis in the figure corresponds to the operator at which the DNN model is sliced, and the resulting performance is evaluated. The stacked bars show the different components of the overall inference time, shown on the left-hand-side y-axis. The dots on the black line mark the frame drop rate for each configuration, with values shown on the right-hand-side y-axis. The split points listed on the x-axis are shown in their execution order. In other words, choosing the split points on the left side means "cutting" the model earlier, and placing fewer operators for processing on the edge. These initial operators of a DNN handle higher-dimensional data, which means that the earlier cuts lead to more time spent in passing metadata between operators. We observed that metadata transmission becomes a bottleneck and results in higher frame drop rate. Looking closely, the split point "pool5" becomes the most efficient deployment with a lowest rate of frame drops. In comparison to simply deploying full DNN models at the edge, as shown in Figure 2c, model slicing improves the overall performance by reducing the frame drop rate by 10%, in this case, and still keeping the same processing delay. Similarly, in comparison to simply deploying full DNN models in the cloud (Figure 2b), model slicing decreases the data movement

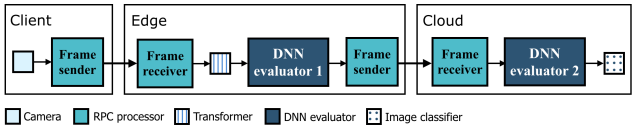


Figure 3: Structure of an image classification pipeline based on Figure 1 when sliced across an edge and a cloud.

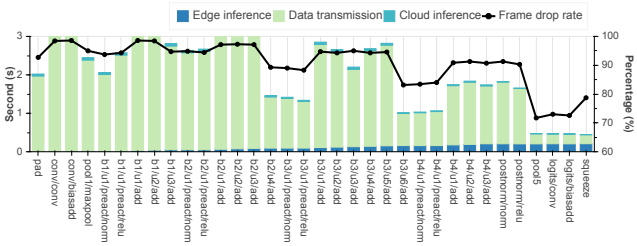


Figure 4: Running image classification with the *ResNet v2 50* model sliced across an edge and a cloud.

cost by requiring lower data transmission between the client and cloud across the Internet. Splitting the DNN at the operator "pool5" reduces the frame drop rate by 25% compared to when running the full DNN in the cloud.

Challenges in slicing a model. The above discussion establishes that the decision on how to best deploy a visual analytics workload at an edge, cloud, or to partition it across both, depends on a number of factors: the DNN model, the workload (frame rate), the configuration of the available resources (compute and the network connectivity). A trivial way to solve this is via a *strongman* approach, i.e., by systematically evaluating splitting the model at each possible operator. But this is only practical when considering how to best deploy a single model for a fixed set of resources (e.g., client and cloud), for a fixed workload (camera resolution and frame rate), and typical LTE or WiFi connectivity. When considering a shared edge infrastructure, such systematic evaluation of all possibilities quickly becomes impractical. Furthermore, the *strongman* methodology is simple when applied to only the simplest types of the current models. A shared edge infrastructure will need to support, potentially concurrently, many applications and the models they use, and it will exhibit variability in the available resources. Additional factors that need to be considered relate to the operations and processing costs associated with the deployment stacks, including the video streaming or deep learning frameworks used by the application. Applying the strongman approach will present scalability challenges and will be of limited practical use.

The Need for Couper. Instead, there is a need for a system that can *efficiently* determine *a good manner* of deploying an application that uses a DNN model and a given edge resource configuration. Preferably, this will be *automated* and will produce *deployment-ready instances* of models and partitions, that can be immediately executed. These requirements led us develop Couper, described next.

4 COUPER OVERVIEW

Assumptions. The design of Couper makes the following assumptions. First is the availability of dynamically provisionable, containerized edge infrastructure, a trend that is prevalent in ongoing edge infrastructure developments. Couper also assumes that application soundness is not affected by running it at the edge vs. in the cloud, i.e., no functional discrepancy arises simply due to running an application at a different location. Next, we assume that information about the structure of the DNN model used in the application is discernible, either as metadata or can be queried using support from the underlying DNN framework which was used to generate the model (e.g., TensorFlow) [1]. Finally, Couper assumes that it has access to minimal staging resources needed to evaluate a workload – i.e., a DNN model – prior to deployment in the edge-cloud production environment. All of the above are reasonable as per the industry trends.

Scope. The scope of Couper is to be able to run a given DNN-based application pipeline across infrastructure consisting of edge and cloud nodes efficiently. Couper does not impact the accuracy or the functional properties of the DNN model or the application (e.g., classification classes, etc.). For slicing, the primary scope of Couper is on system-level support, as opposed to on the development of an optimal algorithm for determining slicing boundaries of an application.

Goals. In order to meet some specified SLAs like latency or quality-of-analysis (frame drops) Couper aims to provide for automated and timely evaluation of different ways to partition a DNN model (i.e., at different split points), to select among them the best slicing point, and to deploy the model partitions for processing across the edge and cloud infrastructure, by using knowledge about the model, the application framework that it uses, and the infrastructure characteristics.

Brief. The high level workflow for Couper is shown in Figure 5. Couper takes as input one (or more) containers that include elements of a streaming visual analytics application pipeline, structural information about the incorporated DNN model and the machine learning framework, workload characteristics in terms of frame rates and resolutions, and infrastructure description concerning the resources available at the edge and the network distances between edge and the cloud.

With these inputs, Couper quickly reduces the search space for the potential slicing boundaries. For instance, we show the benefits of using a user-specified *slicing algorithm* (Subsection § 5.1) which considers the model structure to slice only at articulation points. Couper automatically slices the application, adds the required support for (de)serialization and messaging, and creates a number of candidate container pairs: one for edge and one for cloud. Couper takes those containers and stages their execution with a user-specified or pre-defined workload, hardware configuration and network conditions. In each iteration, Couper selects a pair of candidate as a potential solution or rules it out. This search is short circuited using a user-provided *slice selection method* (Subsection § 5.4) when determined that further execution will not result in selection of a different slice. Finally, when the best pair of containers is selected, they are passed on to the production environment for deployment.

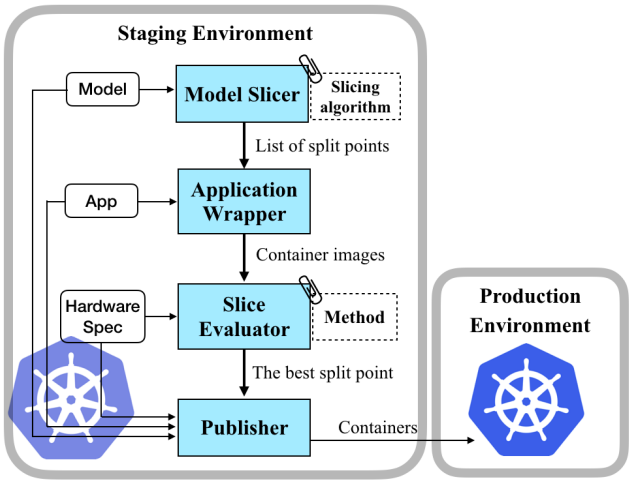


Figure 5: The workflow of Couper

Couper serves two purposes. First, Couper makes it possible to split a model automatically without any support either from the application framework or the ML models. This is made possible due to its unique plug-in-based design that can be used in conjunction with different algorithms. We use Couper to design a number of these algorithms to highlight the trade-offs it affords with respect to finding an ideal slicing point vs. achieving up to 100x faster slicing while still ensuring that the application objectives are met. Second, Couper creates deployment ready slices of DNN models that result in a balanced pipeline for a specific edge configuration and network conditions. This includes support for integration with the underlying streaming and ML frameworks, support to serialize-deserialize the state of the ML models that is exchanged between its elements (operators), and to interface with the container orchestration layer (i.e., Kubernetes). The outcome is novel system-level support for DNN model slicing. The utility of Couper is demonstrated using visual analytics application; we note that it can be applicable to other types of applications that employ DNNs.

Limitations. The implementation of Couper currently supports applications built with the SAF streaming framework, using TensorFlow models, and packaged and deployed as Docker containers. Couper's limitations are only implementation specific.. We do not believe that these are fundamental concerns for the Couper approach. The approach can be ported to other application frameworks, different ML toolsets and packaging technology. In its current form, Couper performs the slicing once for a particular model when the ML application is first started. It considers edge computing resources available at that time, and does not dynamically adapt the slicing within a single run of the same application. However, it is straightforward to see that with a simple automation Couper can be extended to run periodically or on-demand in order to address this limitation. Similarly, the current implementation considers only one edge and one cloud, however, Couper can be extended to multiple tiers of edge infrastructure that may be present or get deployed in the future [51].

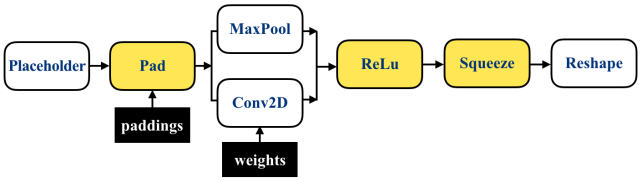


Figure 6: A simple 9-operator model with candidate split points marked in yellow. Three candidates are selected according to Couper's sample slicing algorithm.

5 DESIGN

Couper is designed to work with off-the-shelf production DNN models to quickly find the best split points. Given an application, a DNN and the edge computing resource(s) this application could exploit to optimize for a given performance metric, Couper first evaluates the potentially best deployment for this application in a staging environment; once a satisfactory deployment is determined, it is submitted for use in the production environment.

There are four main components in Couper, illustrated in Figure 5. At the beginning, a **Model Slicer** cuts the DNN based on the user-defined slicing algorithm, and exports the candidate splits. The **Application Wrapper** builds container images for each split partition, based on the pipelined structure of the application and the underlying application frameworks. The **Slice Evaluator** uses a representative workload in Couper as it iterates through the list of candidate splits and evaluates each one. The evaluation is guided by a user-defined method for selecting a best split point. The method specifies which metrics to use when comparing among different configurations, and when to terminate the evaluation process (e.g., until all split points are evaluated, or sooner, as soon as the edge node runs out of computing power to support "bigger" slices). Finally, the **Publisher** prepares the partitions from the selected split point into ready-to-ship containers.

We next describe in more detail each of these components and the input algorithms that control their execution.

5.1 Model Slicer

The Model Slicer module is designed to find all possible split points of the given DNN model. Prior work [39] presented a straightforward approach which considers every operator as a potential split point. However, doing this for production-ready DNN models is non-trivial due to the large number of potential split points. For complex models this would results in many hundreds or even thousands of pipeline configurations to be evaluated. At the very least, this requires time to deploy prepared split points in a staging environment and to process the representative workload through each pipeline configuration, and poses significant resource demand on the infrastructure supporting the evaluation of the workloads.

In contrast, Couper allows for customized slicing algorithms to be specified for the slicing process. We illustrate the utility of this feature by incorporating a sample algorithm that uses the following criteria to reduce the number of possible slicing points which should be considered by Couper.

- **Bypass first and last operators:** Model Slicer does not consider the first and last operators in a model as split points. If one of the two operators is chosen, it simply means that the whole inference is executed on a single machine without any slicing. Moreover, typically the first operator is a pre-processing one that feeds external data into a model, and not really an operation related to inference processing. For example, in a TensorFlow model, this is the *Input* operator.
- **Only split at articulation points:** Model Slicer does not slice at operators where the DNN graph structure is such that it has multiple parallel paths between different operators. It does so to avoid the issues of synchronization that could functionally impact the inference or introduce large amounts of communication overhead at runtime, that would make slicing irrelevant from a performance perspective. We refer to this as the sample algorithm only selects *articulation points* in the DNN as potential split points.
- **Ignore placeholder operators:** While parameters and reading operations in the DNN graph can also be possible split points, these operators should be excluded. If they are selected as split points, the first stage in the pipeline just loads the parameters without performing any operation and then passes them to next stage. As parameters are already part of the pre-trained production DNN, this situation would only result in additional redundant data transmission.

Figure 6 shows an example of a DNN model where the operators selected as possible split points by this algorithm are marked in yellow. This illustrates that the slicing algorithm simplifies the slicing process by selecting fewer candidates to be evaluated by Couper, compared to the list of all possible splits of a DNN.

5.2 Application Wrapper

The **Application Wrapper** creates the corresponding pipeline of containers for the DNN model slices for each of the candidate split points, and passes those to be evaluated by the Slice Evaluator. It packages the application and model slices into Docker [29] images, and configures the range of starting and ending operators corresponding to the split point. Additional intermediate functions are inserted at the head and tail of each slice container to perform (de)serialization of the tensors and data transmission over the network. No other changes are required in the containerized application or the DNN. The serialization is implemented using Protobuf [28] over gRPC [62].

The Application Wrapper manages the buffer sizes of the frame queues in the streaming framework, and the duration of the evaluation, to ensure that the evaluation results are sound. This is a non-intuitive functionality needed by Couper. In ideal scenarios, it should be straightforward to perform the evaluation given a test input workload. However, most visual analytics application pipelines use non-blocking communication and allow for frames to be dropped. This presents a challenge for Couper since it must ensure that enough frames are actually processed during the evaluation in the staging environment. We observed that application behavior converges just in a few frames, so it is sufficient to use a small number of frames for evaluation. However, enough time must be allocated for the pipeline to initialize, process frames and be cleaned up before next iteration.

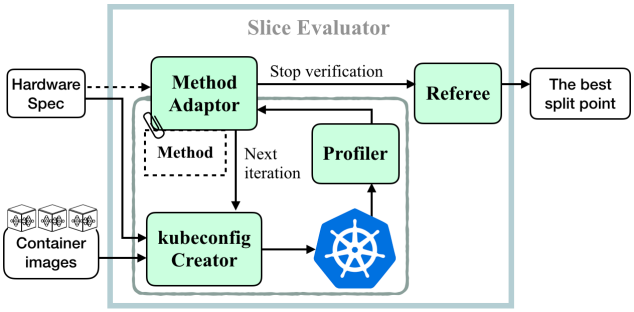
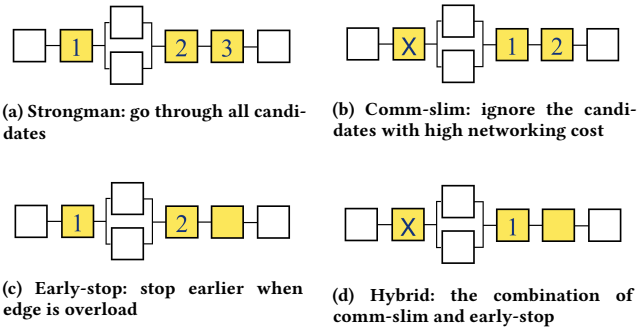


Figure 7: The functions in Slice Evaluator



(a) Strongman: go through all candidates

(b) Comm-slim: ignore the candidates with high networking cost

(c) Early-stop: stop earlier when edge is overload

(d) Hybrid: the combination of comm-slim and early-stop

Figure 8: Illustration of four slice selection methods. The same model example as in Figure 6 is used; the yellow blocks are split candidates selected by Couper's sample slicing algorithm. The blocks are marked with the iteration number of when a candidate is evaluated; *X* indicates candidates which are bypassed by the corresponding selection method.

5.3 Slice Evaluator

The Slice Evaluator evaluates the split points in the candidate list and selects the best one. This procedure is accomplished by four internal modules, illustrated in Figure 7: Method Adaptor, kubeconfig Creator, Profiler and Referee.

The **Method Adaptor** controls the evaluation of the split point configurations based on a user-provided *slice selection method*. The customized method determines (1) whether there are split points which can be eliminated from the candidate list based on network capacity usage, (2) which split point in the candidate list to run, and (3) when to stop. Based on this procedure, in first iteration, Method Adaptor determines whether to remove some split points to reduce the problem space. In each subsequent iteration, Method Adaptor determines whether to stop the evaluation, or to choose a next split point to run.

Since Couper runs on a *Kubernetes* [43] system, it relies on the Kubernetes configuration file, *kubeconfig*, to deploy the pipeline containers across hosts. Each time a split point is chosen by Method Adaptor, the **kubeconfig Creator** updates a new kubeconfig. The kubeconfig file specifies the deployment of pipeline slices to hosts, applies resource limitations on the pipeline containers based on the configuration of the production (or target) edge infrastructure, and configures their communication links to mimic the network conditions. The containers are configured to run as jobs instead of services, since the Slice Evaluator only needs to examine their execution for the fixed duration of the evaluation period. Once a new kubeconfig is created, Creator runs the kubeconfig in the Kubernetes staging area, spawns the containers, and start the execution.

The **Profiler** monitors the container execution, captures relevant per-operator and end-to-end model metrics, low level metrics such as resource utilization, and high level metrics such as processing latency and frame drop rate. When the execution finishes, the Profiler collects the result, sends them back to the Method Adaptor, and the Kubernetes resources created by kubeconfig for the slice point evaluation are cleaned up.

If the Method Adaptor determines that the evaluation process can terminate, the **Referee** goes through each performance log generated by Profiler, and decides the appropriate split point with lowest inference time per frame or lowest data movement cost, as specified by the selection method.

5.4 Sample Slice Selection Methods

Couper permits different methods to be used for the slice selection process. We describe the currently supported ones. This set can be extended with more advanced versions of these algorithms, or in order to account for other important metrics (such as energy).

In order to make a selection, Couper must consider the end-to-end performance of a pipeline, as well as any internal overhead resulting from model slicing. Either factor can be important to the application – i.e., performance vs. operating costs. Also, both factors can lead to reduction of the search space. For instance, if the amount of network communication is crucial (e.g., when an application is deployed to use a cellular network with a limited bandwidth), the split should be chosen among ones which generate low data transfer demand. Similarly, when focusing on end-to-end processing time, if a split results in inference times which exceed a (user-defined) threshold, selection is made from the candidates exercised before it.

A straightforward choice for a method is to take the **strongman approach** (Figure 8a) which iterates over all possible candidate split points and finds the ideal one. Since verifying all possibilities is time-consuming, we identify other methods which significantly reduce the complexity of the evaluation process, while still resulting in model slices which meet the applications' target performance goals. To manage data transmission costs, we devised a **comm-slim method** (Figure 8b) that avoids evaluating the split points with high data transmission demand. The point at which a DNN model is split determines the size of metadata that must be transmitted between the hosts running the DNN slices. The comm-slim method considers metadata size, in conjunction with the network configuration, latency or capacity thresholds, to determine which split points should be bypassed in evaluation. For example, in Figure 8b, the candidate, marked with *X*, is excluded by the comm-slim method due to its large output sizes.

Similarly, to ensure a balanced pipeline execution, we devised another method called **early-stop method** (Figure 8c). It determines when the load generated by a slice at the edge creates a

processing bottleneck in the pipeline, resulting in an increase in dropped frames, or in exceeding the target latency; this information is used for early termination of the evaluation process. During the evaluation loop in the Method Adaptor, the candidate splits points are listed starting with the first operators in the model, thus, the edge node progressively receives more load with each subsequent split point. Using the early-stop method, the evaluation stops when the inference time at the edge stage exceeds the inference time on the cloud. For example, in Figure 8c, one of candidate is not examined, since it may exceed the permissible load on the edge. Compared to the comm-slim method which can be run offline, in the early-stop method, the number of iterations can only be determined at runtime depending on operating conditions. Finally, we consider a method which combines the benefits of both comm-slim and early-stop, and refer to it as the **hybrid method** (Figure 8d). It first reduces the search space by applying the comm-slim method, but then considers only up to a candidate split point which satisfies the early-stop method.

5.5 Publisher

The Publisher's task is to deliver a set of production-ready containers for the visual analytics application. It first builds container images based on the DNN model, the application and the chosen split point. Then, it creates a kubeconfig according to the hardware specification of the production infrastructure. This kubeconfig is ready for production usage to run the containers as services.

6 IMPLEMENTATION

As illustrated on Figure 5, Couper runs on a Kubernetes staging area. We used a Kubernetes cluster to create the staging environment for it provides straightforward controls for container management and for configuration of compute and network resource allocations, necessary to mimic the production environment. The fidelity of the emulation of the production environment in the staging environment is directly related to the accuracy of best split point found by Couper.

DNN model slicing. The Model Slicer and the sample slicing algorithm are implemented in Python with approximately 50 lines of code. They rely on TensorFlow-supported APIs, such as *GraphDef*, *Session.graph.get_operations*, *Operation*'s inputs, type, etc., to load the model, and to understand its graph structure and the types of its operators. Using a single low-end CPU, the Model Slicer requires only few seconds to traverse the graph of the DNN and to generate a list of split point candidates. For NASNet, the DNN model with most operators among the ones we evaluated, the operation takes 3 seconds. For the other DNNs, it completes even faster. Currently, the output list of split candidates is stored as a text file; only a few-kilobyte file can record hundreds of split points.

Packaging the application image. The Application Wrapper and Publisher build container images for the application pipeline. The manner in which application images are built can have a significant impact on the performance and the resource requirements of Couper. The implementation choice influcences the processing time requirements of the Application Wrapper and Publisher. For instance, the use of SAF [65] creates dependencies on several other libraries: for computer vision, such as GStreamer [5] and OpenCV [9]; for machine learning evaluation, such as TensorFlow [10], and for RPC, such as gRPC [62] and Protobuf [28]. The image covering all required libraries is 7GB, which raises challenges when targeting deployments in resource-constrained environments. The image size reaches to 10GB after we packed the SAF source code, image classification application, and all 7 DNN models. The size is further increased by including the CUDA driver and related libraries for GPU execution. However, dynamically creating these container images is also time-consuming, and it can take 2 hours for building the image, depending on the CPU.

In order to eliminate this bottleneck from the Application Wrapper, which needs to repeatedly create or configure container images based on all candidate split points, the implementation of Couper relies on two strategies. The first strategy involves using the same container image across the client, edge, and cloud; the role of each stage is defined by the launch command. Since only a single image is used for the whole pipeline, the image only needs to be rebuilt when there are updates at the application level. The second strategy involves building-in the DNN model's structure and parameters within the application image. For configuring a split point, ideally, it would be needed to modify the DNN model (structure and parameters) and to rebuild the images. To avoid the repeatedly rebuilding, Couper adds interfaces, support from the application or from the the DNN framework, to just attach information describing the operator graph for each DNN slice, as specified by the split point. This information can then be used while booting the container at edge- and cloud-side running the DNN evaluator components. During evaluation, the Application Wrapper attaches the split point configuration at runtime, without rebuilding the images. Once a split point is chosen, the Publisher takes few minutes to pack the DNN slice configurations by referencing the previous base images. We also deploy a private Docker registry [3] with Couper on Kubernetes. It makes sure Couper only needs to build the image once, and can be shipped across hosts.

Evaluating split points. The four functional components in the Slice Evaluator in Couper are run in sequential order. Each component consumes negligible resources for workflow management operations, such as parsing the split candidate list and result logs, generating new kubeconfig files by updating other parameters, and for health-checking the evaluating processes. Most of the computing resources of the staging area are consumed by the pipeline containers being evaluated. During evaluation, for each candidate split point, Method Adaptor sends information to kubeconfig Creator for a new kubeconfig to be created and containers to be spawned. The time requirements of this are also minor comparing to actually running the containers. The containers are run as jobs and they are stopped gracefully (using *SIGTERM*) after few minutes. Currently, the duration is configurable and can be user-provded. As explained earlier, Couper needs to ensure that sufficient number of frames are processed in order to determine the frame drop rate of a split point configuration. Finally, same as creating containers, container cleaning can be performed quickly; as can the tasks performed by the Profiler and Referee, both implemented by lightweight scripts. Note that Couper assumes that the staging area resources should be representative of the resources available in production, since we only rely on controls to scale up and down the container resources, without incorporating more sophisticated performance modeling

	CPU freq (GHz)	#CPU proc	RAM (GB)	GPU	RTT (ms)	
					client	cloud
Client device	2.0	2	1	N/A		
Low-end edge	2.0	4	16		1	65
Mid-end edge	3.1	8	32		15	50
High-end edge	3.1	16	64		25	42
Super-high-end edge	3.1	16	64	1 Nvidia P100	25	42
Cloud server	3.1	48	96	2 Nvidia P100		

Table 4: Hardware specification of the testbed configurations.

and prediction techniques. If GPU machines are used in production deployment, we assume in GPUs are available in the staging area as well.

7 EVALUATION

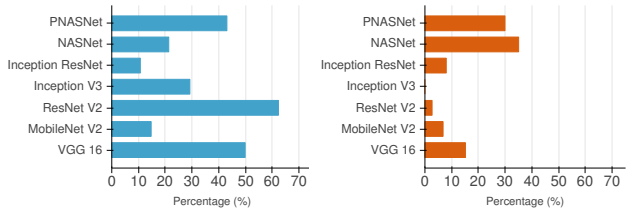
With the evaluation of Couper, we aim to answer to the following questions:

- How do the DNN models sliced with Couper and deployed across edge and cloud improve end-to-end performance compared to full-offloading to cloud?
- How much does Couper improve the time taken to find the best split point compared to a strongman approach?
- How do the methods explored using Couper perform compare to the best split point found by the strongman approach?

Experimental Setup. The experiments were carried out using server-grade machines from the Chameleon [40] research infrastructure. We configured a Kubernetes cluster with three worker machines acting as client, edge and cloud, and a master responsible for orchestration. To model different operating scenarios, comprising different edge node capabilities, we used standard Linux management tools (such as *tc* and *cpufreq-set*) to control parameters such as CPU frequency, number of available cores, use of Nvidia P100 GPUs, and the network between each of those machines. This allowed us to emulate realistic settings of a client-edge-cloud environments, and also to explore different kinds of edge configurations. The hardware configurations used in the presented results are tabulated in Table 4.

Workload. For reproducibility, we used a 1280x720 video as the test query on which analytics are run using different models in different operating scenarios. This video is a commercial film of around 2 minutes. It shows several objects and the correctness of the (sliced) image classification application can easily be verified. The video can be found at Youtube [63]. We used the same video as input data across different models and scenarios. To ensure that Couper works for production-ready models, in all experiments we used official trained models from TensorFlow-Slim [11] without modifying the models or their metadata.

Practical feasibility of model slicing and utility of Couper. We report that Couper was able to slice all production-ready DNN models listed in Table 1, without any modifications to them or the streaming framework SAF [65] that incorporated these models in visual analytic pipelines, and that it was able to improve their end-to-end performance. This also implies the feasibility of model slicing as a practical approach, specifically when resource availability is dynamic and/or limited.



(a) By inference latency **(b) By frame drop rate**

Figure 9: Best performance found by the hybrid method in Couper across different DNNs and edge specifications, normalized by the scenario of running the entire model in the cloud.

The utility of Couper can be gauged by the fact that it can be deployed for arbitrary production-ready DNN models incorporated in other workloads. When deployed for an edge infrastructure, Couper makes it possible for arbitrary workloads to gain benefits from the edge, regardless of the gap in the models' resource requirements and resource availability at the edge.

End-to-end performance. To demonstrate the benefit of Couper on end-to-end performance, we compare the best split points found by the hybrid method of Couper for different edge configurations and network conditions, vs. running the whole inference pipeline in the cloud. The best split point is defined by two SLAs: time-to-inference (Figure 9a) and drop rate of frames (Figure 9b). As evident from the results shown in in Figure 9, for the different models considered in the evaluation, Couper successfully results in 40%-90% improvements in inference latency, and 60%-99% improvement in frame drop rate. We attribute the improvement to reduction in the amount of data that needs to be transferred, and to better utilizing the resource at both the edge and the cloud by partitioning the model. We do not claim that Couper incorporates the best algorithm to find the slicing boundaries, however the results justify the design of Couper and validate its utility. Future work can focus on the design of better algorithms and selection methods.

Note that during the experiments, not all edge configurations were able to fit each of the models. In fact, there is not a single edge configuration that is optimal for all DNN models, owing to their different resource footprints, as discussed earlier in Section § 3. The results in Figure 9 report a summary of the best results obtained with Couper, based on an edge configuration with sufficient processing capacity. For instance, VGG has higher compute requirements so we report the results obtained with a high-end edge, and for NASNet and PNASNet, we report the results from using Couper for a super-high-end edge (the edge with GPU shown in Table 4). The remaining models' requirements can be satisfied with a mid-end edge. This was done to make sure that the results do not have any bias due to resource pressure on the edge due to model execution. *In summary, Couper-enabled model slicing improves end-to-end performance of DNN based visual analytics applications.*

Time-to-slice. A strongman approach to finding the best performant model deployment would entail running every candidate split point provided by the Model Slicer. This could be time consuming, even if we consider a constant time is needed to evaluate one layer.

Method	Original # layer	Strongman	Comm-slim	Hybrid
VGG 16	54	52	20	1
MobileNet V2 1.4	158	155	132	3
ResNet V2 50	205	34	15	1
Inception V3	788	34	15	2
Inception-ResNet-V2	871	106	28	3
NASNet 331	1265	7	3	1
PNASNet 331	939	7	3	1

Table 5: The number of layers taken for evaluation by methods with the setting of client–mid-end edge–cloud

Model	Inception V3	Inception ResNet V2	PNASNet 331
The evaluation time of strongman	> 30	≈ 120	≈ 10
Low-end edge	1	1	1
Mid-end edge	2	3	1
High-end edge	10	16	1

Table 6: The evaluating time by hybrid method across different edges, comparing with strongman method (in minute)

For deeper models, the number of operators could be in thousands. To show how well Couper addresses this, we present experimental evaluations for a fixed hardware specification when using different models. Table 5 lists the original number of operators and the actual split points executed in evaluations, when using the different methods currently supported in Couper. Note that evaluations of each split point are independent of each other, and can be parallelized given enough resources. Effectively, it means that with sufficient resources, the Slice Evaluator can reach a decision faster. Nonetheless, the key point here is not to consider the absolute time spent in evaluation, but to look at the number of points exercised to reach at a decision. It is obvious that verifying more split points will take more time; evaluating all possible ones could take a large amount of time, making it potentially difficult to justify the execution of the slicing evaluation in the first place. The sample algorithms in Couper help mitigate the problem by reducing the search space, first by finding slicable layers, then by choosing which ones among them to evaluate. This reduces the number of split points to be exercised, and it becomes particularly important as DNNs become "deeper" neural network.

Table 5 shows the numbers of points that are not evaluated in Couper. For instance, even when the strongman method is used, it reduces number of points to be evaluated down to 34 candidates from 205 layers for ResNet V2, and also 34 ones from 788 layers for Inception V3. It is a significant reduction in the search space. We posit that these reductions will be even more substantial going forward, as DNNs get deeper. Intuitively, as the complexity of the graph of the models increases, there are fewer articulation points in the DNN model structure (typically represented as a graph) and more dependent dataflows between layers are introduced. As a result, there will be a greater number of operators where slicing would incur more overhead than benefit. When the comm-slim method is used, it considers the network performance to constrain the evaluation. This more aggressively reduces the number of split points to be evaluated, and considers only the ones incurring low data movement costs. The hybrid method further reduces the number of candidate split points, and consequently the evaluation time, based on the early-stop method. For most of the models (VGG, ResNet, NASNet and PNASNet), when considering a "mid-end" edge node (see Table 4) and using the hybrid method, only a single split point is evaluated as a possible splitting candidate. What this simply indicates it that for these models, the cloud is way more powerful than a CPU-only edge node. We show later in this section that this is not a model-specific restriction, instead, for a different edge configuration, when the edge node has a GPU accelerator, slicing across edge and cloud is beneficial. The presence of accelerators is a valid assumption for edge infrastructure components, currently explored by many edge providers.

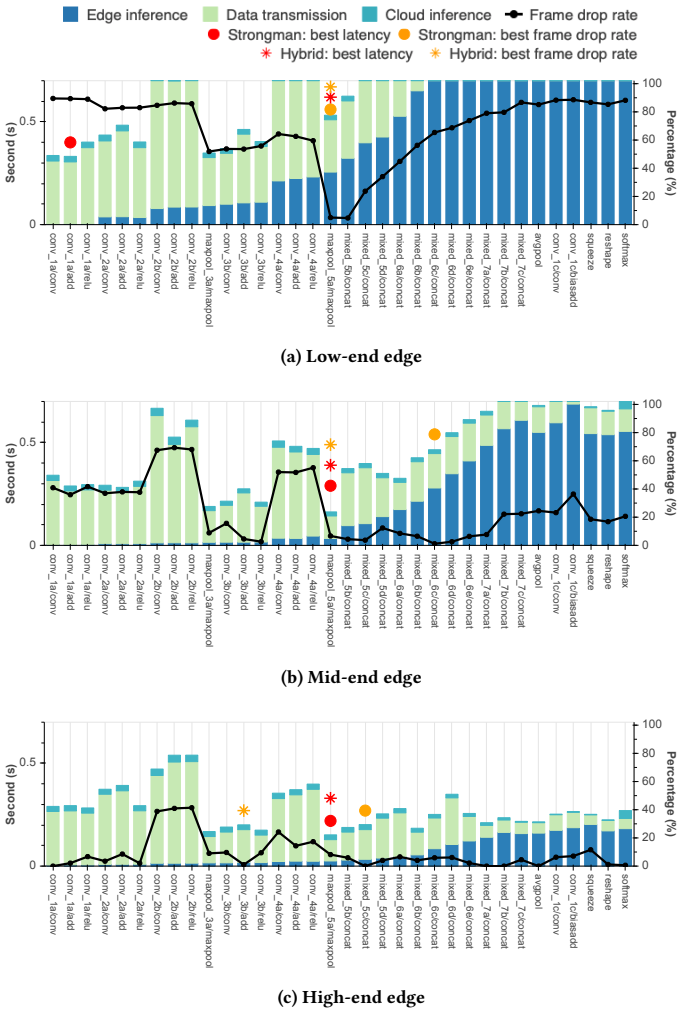
We highlight again that we do not make a claim that Couper incorporates the best methods to slice all future DNN models. It may happen that some breakthrough structural innovation in DNN models changes the interaction between layers. For Couper to remain relevant in those disruptive scenarios, Couper separates the policies from the mechanism for slicing a DNN model, and allows users to provide plug-in methods in the system.

For currently available production-ready models, with the hybrid method, the number of split points to be evaluated not only changes based on the DNN model, but also varies based on the hardware specifications. To demonstrate this, we use Inception V3, Inception ResNet V2 and PNASNet 331, while changing the edge configuration. Table 6 lists the time taken for evaluating slices, in minutes, for the strongman and the hybrid methods. The results show that the hybrid method greatly reduces the evaluation time when compared to strongman. Another observation is that as the compute capacity of the edge increases, the hybrid method tends to try more layers. It is because, first, for the high-end edge setting, where the edge node is closer to the cloud and has lower transmission cost, the comm-slim method explores more layers. Second, the high-end edge is more powerful, therefore it can handle more processing needed during inference, and ends up being terminated later by the early-stop mechanism.

In summary, the results presented in this section show that Couper drastically reduces the time needed to carry out the slicing in any given situation, when compared to a strongman approach.

Quality-of-Slice. Although it is important to reduce the time to slice a DNN, it is also critical to ensure that slicing done in less time also provides performance benefits. To demonstrate that, we used Couper with different client–edge–cloud environments running the same three DNN models as in the last experiment. Figure 10, Figure 11 and Figure 12 show both the frame inference latency and the drop rate per each split point run with the strongman method. The strongman method can choose 106 split points for Inception ResNet V2. For clarity, we only show partial view of all split points or layers, and hide the less critical ones (split point id 28–99).

For each of the three models, we evaluate the outcomes from Couper for each of the three edge configurations listed in Table 4. We use two-colors and two-style symbols in the figures to differentiate the best split points chosen by particular methods and SLAs. Red symbols show the solution with lowest inference latency, while yellow ones show the solutions with smallest frame drop rate; circle symbols mark the best split point found by the strongman method,

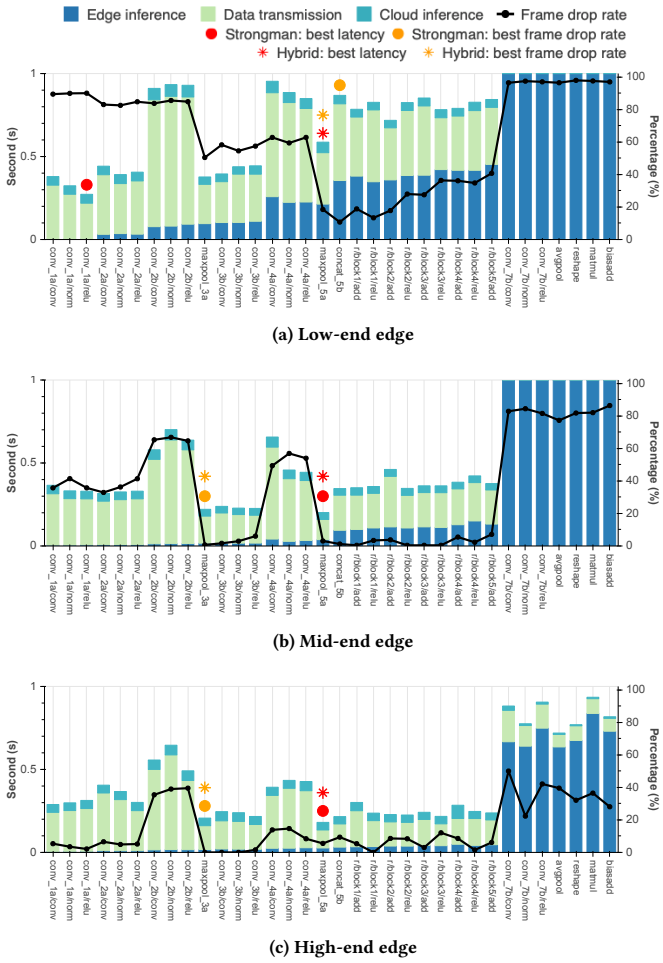


(a) Low-end edge

(b) Mid-end edge

(c) High-end edge

Figure 10: Best split points found by strongman and hybrid methods across different edges with model Inception V3. Red symbols mark the solutions for shortest processing latency; yellow symbols are for shortest frame drop rate.



(a) Low-end edge

(b) Mid-end edge

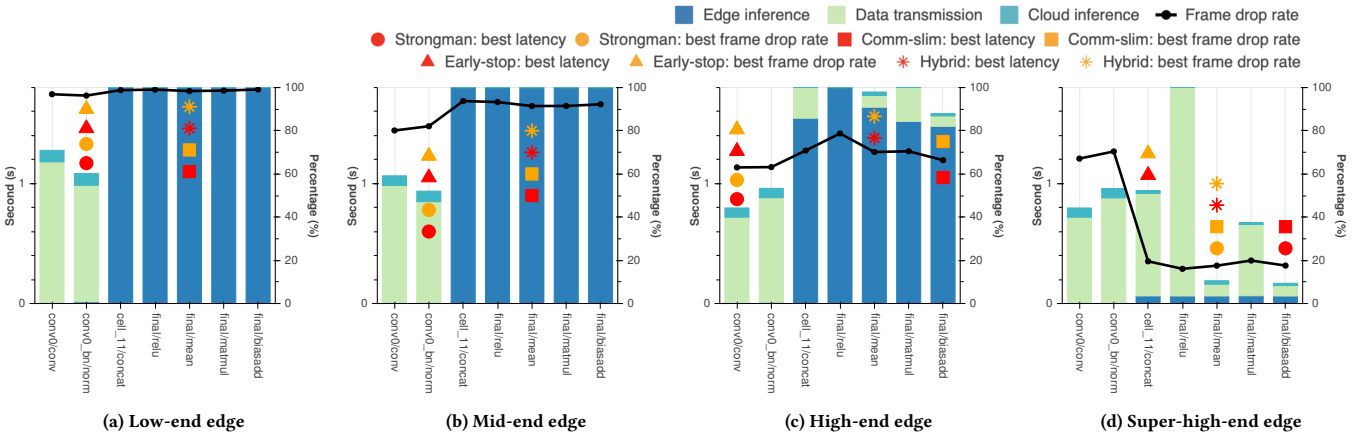
(c) High-end edge

Figure 11: Best split points found by strongman and hybrid methods across different edges with model Inception ResNet V2 (only show split points id 1–27 and 100–106 listed in strongman method). Red symbols mark the solutions for shortest processing latency; yellow symbols are for shortest frame drop rate.

while stars are found by hybrid method. To determine the split points which correspond to the different circles and stars, we use different SLA-specific selection operations in the Referee. In Figure 12, two additional styles of markers represent the other two methods: squares mark the best split points found by comm-slim, and triangles the ones found by early-stop.

In this series of experiments, we can see that the best decision for achieving lowest inference latency may not be the same as the one with smallest frame drop rate. For instance, in Figure 11a, the circle symbol corresponding to the split point for lowest (best) frame drop rate has twice longer (worse) inference latency than the best one of the latency-centric choices. In this scenario, the performance-centric benefits from using the hybrid method are limited; albeit, hybrid still achieves the lowest time-to-slice. This further justifies the design decision to keep methods as plug-ins, and also highlights an opportunity for further research in this space to designing more comprehensive methods for different kinds of use cases, with have different SLAs requirements.

Note, however, that in most cases the hybrid method actually performs very well. When splitting with the goal of lowest inference latency (red symbol), we obtain perfect matches in the case of Figure 10b, Figure 10c, Figure 11b, and Figure 11c. Even when the hybrid method does not find the best split point (which means the star symbols are not marked at the same split point as circle ones), the solution found by the hybrid method still has relatively good performance (such as Figure 10a and Figure 12d). While discussing the position of the best split point in the DNN model, we find that red symbols move to later operators with increases in the edge capacity from low-end edge to high-end edge. At the same



(a) Low-end edge (b) Mid-end edge (c) High-end edge (d) Super-high-end edge

Figure 12: Best split points found by four methods across different edges with model PNASNet 331. Red symbols mark the solutions for shortest processing latency; yellow symbols are for shortest frame drop rate.

time, yellow symbols remain at the same place to maintain a more balanced pipeline. This is interesting and can be attributed to a trade-off between network I/O costs (e.g., latency) and compute costs when deciding on an edge configuration, both of which impact overall inference latency.

With these results, we show that no single type of computing environment is suitable for all models. Inception V3 and Inception ResNet V2 perform well with a mid-end edge (Figure 10b, Figure 11b) are same as for high-end edge, while PNASNet can only perform better with super-high-end edge (Figure 12d).

In the case of PNASNet (Figure 12), we mark in the figures the split point decisions from the comm-slim and early-stop methods as well, to show that the shortcomings of hybrid method can be addressed with different algorithms. In the course of bypassing splits with high networking cost, the hybrid method only moves forward to splits with potentially high inference latency at edge, and ignores the earlier splits which may have comparably shorter overall inference time. Meanwhile, the early-stop method can find the best match within only 3 iterations. The comm-slim method works better than the hybrid method in the super-high-end edge case. Although taking 2 iterations, it can continue to check the later split point and then discover if it becomes the best one for shortest latency.

Summary. In summary, the evaluation reaffirms the possibility that DNN-based ML applications can benefit from edge computing in terms of significantly better performance than when deployed using only the cloud. In addition, they also demonstrate that DNN slicing is practically feasible for unlocking these benefits in terms of faster inference than when running the entire DNN on end-devices or in the cloud. More importantly, the results demonstrate that regardless of the variability or resource limitations at an edge, Couper makes it possible to determine the best split point in a quick and accurate manner.

8 RELATED WORK

Our work leverages a large body of prior research on machine learning for streaming and visual data, on partitioning of machine learning models, and on the role that edge computing can play for accelerating machine learning and analytics services.

Machine learning model improvements and evaluation. When considering neural network improvements, machine learning specialists concentrate on building models which are easier to train and have higher accuracy [34, 58]. Instead of delivering a new, improved model, Wang et al. [66] propose the idea of composing pre-trained models for a faster and still precise inference. Crankshaw et al. [25] present a new framework which links applications and multiple DNNs, in order to find out the most accurate model for a specific application. The goals of Couper are orthogonal; it aims to make it possible to automate and speed up the efficient deployment of a given application-provided model across edge and cloud infrastructure components in a way that best leverages the available resources.

Machine learning on streaming data and model slicing. Running state-of-art models for real-time video analysis increases the accuracy of the analytics, but also the latency and resource consumption. Several prior efforts focus on reducing the overall latency and resource demand of video analytics models by showing the benefits of model partitioning [31, 38, 42] or by developing support for combining and sharing model layers [37]. [31] slices models into short sub-graphs to be executed in parallel on different devices. This solution achieves great performance and improves efficiency, but is hardware-specific and depends on a complicated software stack to optimize the partitioning and to provide for synchronization among the sub-graphs. Other examples present algorithms and methodologies for partitioning models to optimize for resource utilization such as energy constraints [39, 42] or for the characteristics of the network [39]. By providing support for externally-specified slicing and split point selection algorithms, Couper presents a solution which can leverage the decision engines developed in efforts such

as these, while also affording the benefit of automated creation of deployment-ready analytics pipelines.

Machine learning on edge. Machine learning and inference are important workloads for edge computing. In addition to contributions to model slicing for edge [39], other work has made advances in orchestration frameworks and algorithms for deploying and managing machine learning applications in distributed, multi-tenant and/or multi-device scenarios [59–61], for determining the best model for a given edge [48], or for specializing machine learning for edge scenarios [26]. Prior research has also shown the utility of using machine learning to optimize control-plane operation and other functionality performed at the edge, such as security, use of storage or network capacity [18, 49]. These efforts illustrate the growing trend of building DNN-enabled applications for the edge, and the need for a systems such as Couper which can assist with their deployment across diverse and shared edge infrastructure.

9 CONCLUSIONS

We have presented Couper, a tool to automatically slice DNN based visual analytics applications to run them efficiently at the edge infrastructure. We showed that Couper can be applied with arbitrary production models and for different infrastructure configurations. Couper integrates the basic mechanisms needed for its operation, but provides support for different decision engines, in the form of algorithms for model slicing and for selection of best model slices. This, in turn, allows Couper users to trade the overheads incurred in terms of execution time and resources needed to decide how to split a model, vs. the quality of the model slices with respect to the target metrics. We demonstrate experimentally that Couper can automatically create deployment-ready DNN pipelines which can deliver significant improvements in inference time (i.e., latency) or inference quality (i.e., percentage of processed frames), with overheads which range from comparable to 100x less than a strongman approach. We believe that Couper will be helpful in deployment of DNN-based applications at edge computing infrastructure in general.

REFERENCES

[1] [n. d.]. A Tool Developer's Guide to TensorFlow Model Files. https://www.tensorflow.org/guide/extend/model_files

[2] [n. d.]. AirBux Aerial: Drone based visial data collection and delivery. https://airbusaerial.com/.

[3] [n. d.]. Docker Registry. https://docs.docker.com/registry/

[4] [n. d.]. Fog Computing. http://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf.

[5] [n. d.]. GStreamer: open source multimedia framework. https://gstreamer.freedesktop.org

[6] [n. d.]. Intel Network Edge Virtualization. https://networkbuilders.intel.com/network-technologies/nev.

[7] [n. d.]. Models and layers | TensorFlow. https://www.tensorflow.org/js/guide/models_and_layers

[8] [n. d.]. Netflix Open Connect. https://openconnect.netflix.com/en/.

[9] [n. d.]. OpenCV. https://opencv.org

[10] [n. d.]. TensorFlow. https://www.tensorflow.org.

[11] [n. d.]. TensorFlow-Slim image classification model library. https://github.com/tensorflow/models/tree/master/research/slim

[12] [n. d.]. Vapor. https://www.vapor.io

[13] 2018. Akraino Edge Stack. https://www.lfedge.org/projects/akraino/.

[14] 2019. Deutsche Telekom Completes World's First Public Mobile Edge Network, Powered By MobiledgeX Edge-Cloud R1.0. http://mobiledgex.com/press-releases/2019/02/19/deutsche-telekom-completes-worlds-first-public-mobile-edge-network-powered-by-mobiledgex-edge-cloud-r1-0.

[15] Telefonica Alpha. 2017. The Edge: Evolution or Revolution? http://acm-ieee-sec.org/2017/Edge%20Computing%20SEC%20Keynote%20Oct%202017%20Pablo%20Rodriguez.pdf

[16] AWS. 2018. AWS Greengrass - Amazon Web Services. https://aws.amazon.com/greengrass/ [Online].

[17] Microsoft Azure. 2018. IoT Edge | Microsoft Azure. https://azure.microsoft.com/services/iot-edge/ [Online].

[18] Ketan Bhardwaj, Joaquin Chang Miranda, and Ada Gavrilovska. 2018. Towards IoT-DDOS Prevention Using Edge Computing. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge'18)*.

[19] Ketan Bhardwaj, Ming-Wei Shih, Pragya Agarwal, Ada Gavrilovska, Taesoo Kim, and Karsten Schwan. 2016. Fast, Scalable and Secure Onloading of Edge Functions using AirBox. In *Proceedings of the 1st IEEE/ACM Symposium on Edge Computing (SEC'16)*.

[20] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog Computing and Its Role in the Internet of Things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing (MCC '12)*. ACM, New York, NY, USA, 13–16.

[21] Charles E. Catlett, Peter H. Beckman, Rajesh Sankaran, and Kate Kusiak Galvin. 2017. Array of things: a scientific research instrument in the public way: platform design and early lessons learned. In *Proceedings of the 2nd International Workshop on Science of Smart City Operations and Platforms Engineering, SCOPE@CPSWeek 2017, Pittsburgh, PA, USA, April 21, 2017*. 26–33. https://doi.org/10.1145/3063386.3063771

[22] Cisco. 2017. Cisco Visual Networking Index: Forecast and Methodology, 2016–2021. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html

[23] Google Cloud. 2018. Cloud IoT Core | Google Cloud. https://cloud.google.com/iot-core/ [Online].

[24] Zeromq community. 2011. Distributed messaging - zeromq. http://zeromq.org

[25] Daniel Crankshaw, Xin Wang, Giulio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. 2017. Clipper: A Low-Latency Online Prediction Serving System. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*. 613–627.

[26] Harshit Daga, Patrick Nicholson, Ada Gavrilovska, and Diego Lugones. 2019. Cartel: A System for Collaborative Transfer Learning at the Edge Cloud. In *ACM Symposium on Cloud Computing (SoCC'19)*.

[27] Jim Davis, Philbert Shih, and Alex Marcham. 2018. State of the Edge: A Market and Ecosystem Report for Edge Computing. https://www.stateoftheedge.com.

[28] Google developers. 2018. Protocol Buffers. https://developers.google.com/protocol-buffers/

[29] Docker. 2018. Docker - Build, Ship, and Run Any App, Anywhere. https://www.docker.com

[30] ETSI Mobile Edge Computing [n. d.]. ETSI Mobile Edge Computing. http://goo.gl/Qef61X.

[31] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, Stephen Heil, Prerak Patel, Adam Sapek, Gabriel Weisz, Lisa Woods, Sitaram Lanka, Steven K. Reinhardt, Adrian M. Caulfield, Eric S. Chung, and Doug Burger. 2018. A Configurable Cloud-Scale DNN Processor for Real-Time AI. In *45th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2018, Los Angeles, CA, USA, June 1-6, 2018*. 1–14.

[32] Harshit Gupta, Zhuangdi Xu, and Umakishore Ramachandran. 2018. DataFog: Towards a Holistic Data Management Platform for the IoT Age at the Network Edge. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*. USENIX Association, Boston, MA. https://www.usenix.org/conference/hotedge18/presentation/gupta

[33] Kiryong Ha, Zhuo Chen, Wenlu Hu, Wolfgang Richter, Padmanabhan Pillai, and Mahadev Satyanarayanan. 2014. Towards wearable cognitive assistance. In *The 12th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys'14, Bretton Woods, NH, USA, June 16-19, 2014*. 68–81.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition*. CVPR, Las Vegas, NV, USA, 770–778.

[35] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R. Ganger, Phillip B. Gibbons, and Onur Mutlu. 2017. Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*. 629–647.

[36] Vodafone Qatar's Chief Operating Officer Mohamed Al Sadah Interview. [n. d.]. Vodafone Smart Stadium. http://www.businessreviewweurope.eu/technology/1116/Vodafone-smart-stadiums-and-the-2022-Qatar-World-Cup.

[37] Angela H. Jiang, Daniel L.-K. Wong, Christopher Canel, Lilia Tang, Ishan Misra, Michael Kaminsky, Michael A. Kozuch, Padmanabhan Pillai, David G. Andersen, and Gregory R. Ganger. 2018. Mainstream: Dynamic Stem-Sharing for Multi-Tenant Video Processing. In *2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018*. 29–42.

[38] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: Optimizing Deep CNN-Based Queries over Video Streams at Scale. *PVLDB* 10, 11 (2017), 1586–1597.

[39] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor N. Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS)*. ASPLOS, China, 615–629.

[40] Kate Keahey, Pierre Riteau, Dan Stanzione, Tim Cockerill, Joe Mambretti, Paul Rad, and Paul Ruth. 2018. Chameleon: a Scalable Production Testbed for Computer Science Research. In *Contemporary High Performance Computing: From Petascale toward Exascale* (1 ed.), Jeffrey Vetter (Ed.). Chapman & Hall/CRC Computational Science, Vol. 3. CRC Press, Boca Raton, FL, Chapter 5.

[41] Jong Hwan Ko, Taesik Na, Mohammad Faisal Amir, and Saibal Mukhopadhyay. 2018. Edge-Host Partitioning of Deep Neural Networks with Feature Space Encoding for Resource-Constrained Internet-of-Things Platforms. In *15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*.

[42] Jong Hwan Ko, Taesik Na, Mohammad Faisal Amir, and Saibal Mukhopadhyay. 2018. Edge-Host Partitioning of Deep Neural Networks with Feature Space Encoding for Resource-Constrained Internet-of-Things Platforms. *CoRR* abs/1802.03835 (2018).

[43] Kubernetes. 2018. Production-Grade Container Orchestration. https://kubernetes.io

[44] Edward A. Lee, Björn Hartmann, John Kubiatowicz, Tajana Simunic Rosing, John Wawrzynek, David Wessel, Jan M. Rabaey, Kris Pister, Alberto L. Sangiovanni-Vincentelli, Sanjit A. Seshia, David Blaauw, Prabal Dutta, Kevin Fu, Carlos Guestrin, Ben Taskar, Roozbeh Jafari, Douglas L. Jones, Vijay Kumar, Rahul Mangharam, George J. Pappas, Richard M. Murray, and Anthony Rowe. 2014. The Swarm at the Edge of the Cloud. *IEEE Design & Test* 31, 3 (2014), 8–20.

[45] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled Ben Letaief. 2017. A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Communications Surveys and Tutorials* 19, 4 (2017), 2322–2358. https://doi.org/10.1109/COMST.2017.2745201

[46] Rick Merritt. 2018. AI Becomes the New Moore's Law. https://www.eetimes.com/document.asp?doc_id=1333471.

[47] Iain Morris. 2019. DT-Owned MobiledgeX to Power German Telco's Edge Rollout. https://www.lightreading.com/mobile/mec-(mobile-edge-computing)/dt-owned-mobiledgex-to-power-german-telcos-edge-rollout/d/d-id/740845.

[48] Samuel S. Ogden and Tian Guo. 2018. MODI: Mobile Deep Inference Made Efficient by Edge Computing. In *USENIX Workshop on Hot Topics in Edge Computing, HotEdge 2018, Boston, MA, July 10, 2018.*

[49] Arun Ravindran and Anjus George. 2018. An Edge Datastore Architecture For Latency-Critical Distributed Machine Vision Applications. In *USENIX Workshop on Hot Topics in Edge Computing, HotEdge 2018, Boston, MA, July 10, 2018.*

[50] Ananda Samajdar, Parth Mannan, Kartikay Garg, and Tushar Krishna. 2018. GeneSys: Enabling Continuous Learning through Neural Network Evolution in Hardware. In *Proc of 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'18).*

[51] Mahadev Satyanarayanan, Wei Gao, and Brandon Lucia. 2019. The Computing Landscape of the 21st Century. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications (HotMobile '19).*

[52] Alisha Seam. 2019. AT&T Unlocks the Power of Edge Computing: Delivering Interactive VR over 5G. https://about.att.com/innovationblog/2019/02/edge_computing_vr.html.

[53] Hardik Sharma, Jongse Park, Divya Mahajan, Joon Kyung Kim, Chenkai Shao, Asit Mishra, Emmanuel Amaro, and Hadi Esmaeilzadeh. 2016. From High-Level Deep Neural Models to FPGAs. In *IEEE/ACM International Symposium on Microarchitecture (MICRO'16).*

[54] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646.

[55] Rags Srinivasan and Agnieszka Zielinska. 2019. Data at the Edge. https://www.stateoftheedge.com.

[56] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. 2016. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *CoRR* abs/1602.07261 (2016).

[57] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. 1–9.

[58] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2016. Rethinking the Inception Architecture for Computer Vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. 2818–2826.

[59] Nisha Talagala, Swaminathan Sundararaman, Vinay Sridhar, Dulcardo Arteaga, Qianmei Luo, Sriram Subramanian, Sindhu Ghanta, Lior Khermosh, and Drew Roselli. 2018. ECO: Harmonizing Edge and Cloud with ML/DL Orchestration. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*. USENIX Association, Boston, MA.

[60] Zeyi Tao and Qun Li. 2018. eSGD: Communication Efficient Distributed Deep Learning on the Edge. In *USENIX Workshop on Hot Topics in Edge Computing, HotEdge 2018, Boston, MA, July 10, 2018.*

[61] Surat Teerapittayanon, Bradley McDanel, and H.T. Kung. 2017. Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices. In *IEEE 37th International Conference on Distributed Computing Systems (ICDCS).*

[62] CNCF project The gPRC authors. 2019. grpc.io. https://grpc.io

[63] Tile. 2017. tile | Lost panda | together we find. https://www.youtube.com/watch?v=u-evd9B-ZKg

[64] Deepak Vasisht, Zerina Kapetanovic, Jongho Won, Xinxin Jin, Ranveer Chandra, Sudipta Sinha, Ashish Kapoor, Madhusudhan Sudarshan, and Sean Stratman. 2017. FarmBeats: An IoT Platform for Data-Driven Agriculture. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17).*

[65] viscloud. 2018. Streaming Analytics Framework (SAF). https://github.com/viscloud/saf

[66] Xin Wang, Yujia Luo, Daniel Crankshaw, Alexey Tumanov, and Joseph E. Gonzalez. 2017. IDK Cascades: Fast Deep Learning by Learning not to Overthink. *CoRR* abs/1706.00885 (2017).

[67] Dale F. Willis, Arkodeb Dasgupta, and Suman Banerjee. 2014. ParaDrop: A Multi-tenant Platform for Dynamically Installed Third Party Services on Home Gateways. In *Proceedings of the 2014 ACM SIGCOMM Workshop on Distributed Cloud Computing (DCC '14)*. ACM, New York, NY, USA, 43–44. https://doi.org/10.1145/2627566.2627583

[68] Tan Zhang, Aakanksha Chowdhery, Paramvir (Victor) Bahl, Kyle Jamieson, and Suman Banerjee. 2015. The Design and Implementation of a Wireless Video Surveillance System. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking (MobiCom '15)*. ACM, New York, NY, USA, 426–438. https://doi.org/10.1145/2789168.2790123