GEMEL: Model Merging for Memory-Efficient, Real-Time Video Analytics at the Edge

Arthi Padmanabhan* Neil Agarwal¶ Anand Iyer† Ganesh Ananthanarayanan†
Yuanchao Shu† Nikolaos Karianakis† Guoqing Harry Xu* Ravi Netravali¶

*UCLA †Microsoft Research ¶Princeton University

Abstract

Video analytics pipelines have steadily shifted to edge deployments to reduce bandwidth overheads and privacy violations, but in doing so, face an ever-growing resource tension. Most notably, edge-box GPUs lack the memory needed to concurrently house the growing number of (increasingly complex) models for real-time inference. Unfortunately, existing solutions that rely on time/space sharing of GPU resources are insufficient as the required swapping delays result in unacceptable frame drops and accuracy violations. We present *model merging*, a new memory management technique that exploits architectural similarities between edge vision models by judiciously sharing their layers (including weights) to reduce workload memory costs and swapping delays. Our system, GEMEL, efficiently integrates merging into existing pipelines by (1) leveraging several guiding observations about per-model memory usage and inter-layer dependencies to quickly identify fruitful and accuracy-preserving merging configurations, and (2) altering edge inference schedules to maximize merging benefits. Experiments across diverse workloads reveal that GEMEL reduces memory usage by up to 60.7%, and improves overall accuracy by 8-39% relative to time/space sharing alone.

1 Introduction

Fueled by the proliferation of camera deployments and significant advances in deep neural networks (DNNs) for vision processing (e.g., classification, detection) [19, 29, 43, 61, 65], live video analytics have rapidly grown in popularity [24, 35, 53, 63, 97]. Indeed, major cities and organizations around the world now employ thousands of cameras to monitor intersections, homes, retail spaces, factories, and more [1, 5, 6, 9]. The generated video feeds are continuously and automatically queried using DNNs to power long-running applications for autonomous driving, footfall tracking, traffic coordination, business analytics, and surveillance [2, 10–12, 34].

In order to deliver highly-accurate query responses in real time, video analytics deployments have steadily migrated to the edge [24, 69, 91]. More specifically, pipelines routinely incorporate *on-premise* edge servers (e.g., Microsoft EdgeBox [4], Amazon OutPosts [3]) that run in hyper-proximity to cameras (in contrast to traditional edge servers [33, 37, 70, 89]), and possess on-board GPUs to aid video processing. These *edge boxes* are used to complement (or even replace [27, 30]) distant cloud servers by locally performing as many inference tasks on live video streams as possible [30, 47, 63, 101]. Generating query responses directly on edge boxes reduces transfer delays for shipping data-dense video over wireless links [30, 41, 64, 102] while also bringing resilience to outbound edge-network link failures [7, 71] and compliance with regional data privacy restrictions [68, 74].

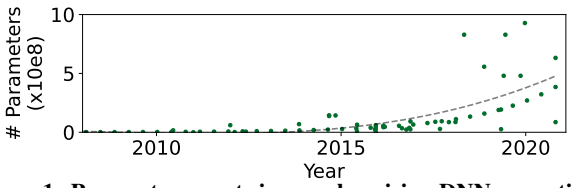

Figure 1: Parameter counts in popular vision DNNs over time. Data drawn from [80].

To reap the above benefits, video analytics deployments must operate under the limited computation resources offered by edge boxes. On the one hand, due to cost, power, and space constraints, edge boxes typically possess weaker GPUs than their cloud counterparts [4, 27, 83]. On the other hand, analytics deployments face rapidly increasing workloads due to the following trends: (1) more camera feeds to analyze [27, 47, 49], (2) more models to run due to increased popularity and shifts to bring-your-own-model platforms [16, 23, 38, 48], and (3) increased model complexity, primarily through growing numbers of layers and parameters (Figure 1) [15, 50, 51, 92]. Taken together, the result is an ever-worsening resource picture for edge video analytics.

Problems. Although GPU computation resources are holistically constrained on edge boxes, this paper focuses on *GPU memory restrictions*, which have become a primary bottleneck in edge video analytics for three main reasons. First, GPU memory is costly due to its high-bandwidth nature [72, 75, 81], and is thus unlikely to keep pace with the ever-growing memory needs of DNNs (Figure 1). Second, we empirically find that existing memory management techniques that time/space-share GPU resources [25, 39, 45, 50, 82, 94] are insufficient for edge video analytics, resulting in skipped processing on 19-84% of frames, and corresponding accuracy drops up to 43% (§3). The underlying reason is that the costs of loading vision DNNs into GPU memory (i.e., swapping) are prohibitive and often exceed the corresponding inference times, leading to sub-frame-rate ($<$ 30 fps) processing and dropped frames due to SLA violations [82, 98]. Such accuracy drops are unacceptable for important vision tasks, especially given that each generation of vision DNNs brings only 2-10% of accuracy boosts – that after painstaking tuning [21, 46, 57, 85]. Third, in comparison with computation bottlenecks [30, 39, 40, 53, 63], GPU memory restrictions during inference have been far less explored in video analytics.

Contributions. We tackle this memory challenge by making two main contributions described below. The design and evaluation of our solution are based on a wide range of popular vision DNNs, tasks, videos, and resource settings that reflect workloads observed in both our own multi-city pilot video analytics deployment and in prior studies (§2).

Our first contribution is *model merging*, a fundamentally new approach to tackling GPU memory bottlenecks in edge video analytics that is complementary to time/space-sharing strategies (§4). With merging, we aim to share *architecturally identical* layers across the models in a workload such that only one copy of each shared layer (i.e., one set of weights) must be loaded into GPU memory for all models that include it. In doing so, merging reduces both the number of swaps required to run a workload (by reducing the overall memory footprint) and the cost of each swap (by lowering the amount of new data to load into GPU memory).

Our merging approach is motivated by our (surprising) finding that vision DNNs share substantial numbers of layers that are architecturally (i.e., excluding weights) identical (§4.1). Such commonalities arise not only between identical models (100% sharing), but also across model variants in the same (up to 25.3%) and in different (up to 92.3%) families. The reason is that, despite their (potentially) different goals, vision DNNs ultimately employ traditional CV operations (e.g., convolutions) [21,57], operate on unified input formats (e.g., raw frames), and perform object-centric tasks (e.g., detection, classification) that rely on common features such as edges, corners, and motion [28, 32, 58, 59, 77, 90, 103, 104].

Our analysis reveals that exploiting these architectural commonalities via merging has the potential to substantially lower memory usage (17.9-86.4%) and boost accuracy (by up to 50%) in practice. However, achieving those benefits is complicated by the fact that edge vision models typically use different weights for common layers due to training non-linearities [55, 56] and variance in target tasks, objects, and videos;[1] and yet, merging requires using unified weights for each shared layer. Digging deeper, we observe that there exists an *inverse relationship* between the number of shared layers and achieved accuracy during retraining. Intuitively, this is because for shared layers to use unified weights, other layers must adjust their weights accordingly during retraining; the more layers shared, the harder it is for (the fewer) other layers to find weights to accommodate such constraints and successfully learn the target functions [22, 62]. Worse, determining the right layers to merge is further complicated by the fact that (1) it is difficult to predict precisely how many layers will be shareable before accuracy violations occur, and (2) each instance of retraining is costly.

Our second contribution is GEMEL,[2] an end-to-end system that practically incorporates model merging into edge video analytics by automatically finding and exploiting merging opportunities across user-registered vision DNNs (§5). GEMEL tackles the above challenges by leveraging two key observations: (1) vision DNNs routinely exhibit power-law distributions whereby a small percentage of layers, often towards the end of a model, account for most of the model's memory usage, and (2) merging decisions are agnostic to inter-layer dependencies, and accordingly, a layer's mergeability does not improve if other layers are also shared.

Building on these observations, GEMEL follows an *incremental* merging process whereby it attempts to share one additional layer during each iteration, and selects new layers in a memory-forward manner, i.e., prioritizing the (few) memory-heavy layers. In essence, this approach aims to reap most of the potential memory savings as quickly, and with as few shared layers, as possible. GEMEL further accelerates the merging process by taking an adaptive approach to retraining that detects and leverages signs of early successes and failures. At the end of each successful iteration, GEMEL ships the resulting merged models to the appropriate edge servers, and carefully alters the time/space-sharing scheduler – a merging-aware variant of Nexus [82] in our implementation – to maximize merging benefits, i.e., by organizing merged models to reduce the number of swaps, and the delay for each one. Importantly, GEMEL verifies that merging configurations meet the required per-model accuracy targets *prior* to deployment at the edge, and also periodically tracks the presence of data drift for those models.

Results. We evaluated GEMEL on a wide range of workloads and edge settings (§2). Overall, GEMEL reduces memory requirements by up to 60.7%, which is 5.9-52.3% more than stem-sharing approaches that are fundamentally restricted to sharing contiguous layers from the start of models (Mainstream [52]), and within 9.3-29.0% of the theoretical maximum savings (that disregard layer weights). These memory savings lead to 13-44% fewer skipped frames and overall accuracy improvements of 8-39% compared to space/time-sharing GPU schedulers alone (Nexus [82]).

2 Methodology & Pilot Study

We begin by describing the workloads used in this paper. These workloads were derived from our experience in deploying a pilot video analytics system in collaboration with two major US cities, one from each coast, for road traffic monitoring. To ensure that our analysis is representative of current and future video analytics deployments, we constructed a series of workloads that reflect the range of vision DNNs, tasks, video properties, and edge resources seen in both our pilot deployment and prior studies. We describe each axis, before explaining the construction of workloads.

Models and tasks. We selected the 7 most popular families of models across our pilot deployment and the recent literature [26, 27, 44, 45, 47, 52–54, 63, 93]: YOLO, Faster RCNN, ResNet, VGG, SSD, Inception, and Mobilenet. From each

[1] This limits the applicability of prior sharing strategies such as Pretzel [60] that rely on shared layers to have the same weights (§7).

[2] A gemel is a pair of trees whose branches have fused together over time.

family, we selected up to 4 model variants (if available) that exhibit different degrees of complexity and compression. For instance, from the YOLO family, we consider YOLOv3 and Tiny YOLOv3; similarly, from the ResNet family, we consider ResNet{18, 50, 101, 152}. The selected models focus on two tasks – object classification and detection – and for each, we train different versions for two objects of interest: people and vehicles. Classification and detection accuracy are measured using standard F1 and mAP scores [36].

Videos. Our dataset consists of video streams from 12 different cameras in our pilot deployment that span two metropolitan areas. From each region, we consider cameras located at adjacent intersections, as well as cameras spaced farther apart within the same metropolitan area; doing so enables us to consider different edge box placements, i.e., those placed directly at a traffic intersection, or those placed further upstream to service a slightly larger geographic location. From each stream, we scraped 120 minutes of video that cover 24-hour periods from four times of the year. Thus, our dataset exhibits varying levels of lighting, weather patterns, and object densities. The streams also vary in terms of camera orientation (relative to the scenes), frame rate, and resolution.

Edge boxes. Our review of on-premise edge boxes focused on five commercial offerings: Microsoft EdgeBox [4], Amazon OutPosts [3], Sony REA-C100 [84], Nvidia Jetson [8], and Hailo Edge-AI-box [14]. These servers each possess onboard GPUs and offer between 2-16 GB of total GPU memory. Since edge inferences do not typically span multiple GPUs, we focus on model merging and inference scheduling *per GPU*. Note that this does not restrict GEMEL to single GPU settings; rather, it means that our merging and scheduling techniques are applied separately to the DNNs in each GPU, with the assumption that each merged model is scheduled to run on one GPU. We defer an exploration of model parallelism for merged models to future work.

Workload construction. Recent works highlight that tens of video feeds are typically routed to each on-premise edge box [12, 47], which runs upwards of 10 queries (or DNNs) on each feed [16, 27]. Our experience reflects similar requirements: it was typical to direct the maximum possible number of feeds to an edge box, with the goal of *minimizing the number of edge boxes required to process the workload*. To cover this space, and since we focus on per-GPU inference optimization, we generated an exhaustive list of all possible workloads sized between 2-20 DNNs using the models above. We then sorted this list in terms of the potential memory savings (using the methodology in §4), and selected 15 workloads: 3 random workloads from the lower quartile (i.e., *Low Potential (L1-3)*), 6 from the middle 50% (i.e., *Medium Potential (M1-6)*), and 6 from the upper quartile (i.e., *High Potential (H1-6)*). Each workload was randomly assigned to one of the cities, with the constituent models being randomly paired with the available video feeds and an accuracy target in {90, 95, 99}%. §A details the workloads, each of which exhibits heterogeneity in terms of the families, tasks, videos, and target objects of the included models.

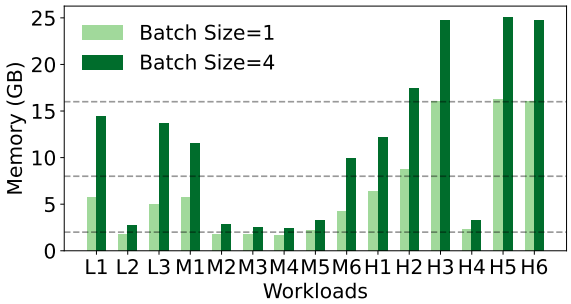

Figure 2: Per-workload memory requirements for two popular batch sizes used in video analytics [82]. Dashed lines represent the available GPU memory on several commercial edge boxes.

Note on result presentation. Results on end-to-end accuracy are dependent on the available GPU memory. However, each workload requires a different minimum amount of memory to run, i.e., the GPU should be able to load and run the most memory-intensive model in isolation for a batch size of 1. Further, the maximum memory needed to avoid swapping (i.e., to load all models and run one at a time) also varies per workload. To ensure comparability across all presented accuracy results and to maintain focus on memory-bottlenecked scenarios, we assign each workload three memory settings to be evaluated on: (1) the minimum value (*min*), (2) 50% of the max value (*50%*), and (3) 75% of the max value (*75%*). §A lists these values per workload.

3 Motivation

In this section, we first characterize the ever-pressing memory tension facing edge video analytics (§3.1), and then explain the limitations of existing memory management solutions in addressing that tension (§3.2).

3.1 Memory Pressure in Edge Video Analytics

To run inference with a given model, that model's layers and parameters must be loaded into the GPU's memory, with sufficient space reserved to house intermediate data generated while running, e.g., activations. The amount of data generated (and thus, memory consumed) during inference depends on both the model architecture and the batch size used; a higher batch size typically requires more memory.

Figure 2 shows the total amount of memory (i.e., for both loading and running) required for each of our workloads and two batch sizes; note that the listed numbers exclude the fixed memory that machine learning frameworks reserve for operation, e.g., PyTorch [18] consumes 0.8 GB by default. As shown, many workloads do not directly fit into edge box GPUs, and the number of edge boxes necessary to support a given workload can quickly escalate. For instance, even with a batch size of 1 frame, 73% of our workloads require more than one edge box possessing 2 GB of GPU memory; with

Model	Load (GB)	Run (GB)		
		BS=1	BS=2	BS=4
YOLOv3	0.242	0.518	0.728	1.22
ResNet152	0.244	0.648	0.978	1.71
ResNet50	0.118	0.346	0.498	0.838
VGG16	0.536	0.738	0.890	1.18
Tiny YOLOv3	0.042	0.152	0.180	0.238
Faster RCNN	0.732	3.70	6.96	12.47
Inceptionv3	0.120	0.190	0.228	0.340
SSD-VGG	0.106	0.230	0.328	0.506

Table 1: Memory requirements (in GB) for loading and running inference for various models and three different batch sizes (in frames). Run values include load values, but exclude memory requirements of serving frameworks, e.g., PyTorch.

a batch size of 4, 60% and 27% require more than one edge box with 8 GB and 16 GB of memory, respectively.

Table 1 breaks this memory pressure down further by listing the amount of loading and running memory required for representative models in our workloads. When analyzed in the context of the scale of edge video analytics workloads, the picture is bleak, even with a batch size of 1. For example, a 2 GB edge box can support only 1, 2, or 3 VGG16, YOLOv3, or ResNet50 models, respectively, after accounting for the memory needs of the serving framework. Moving up to 8 and 16 GB edge boxes (of course) helps, but not enough, e.g., an 8 GB box can support 13 YOLOv3 or 2 Faster RCNN models, both of which are a drastic drop from the 10s of models that workloads already involve (§2).

3.2 Existing GPU Memory Management Solutions

Existing solutions to handling the memory restrictions described above either aim to share the available GPU memory in space or time, or reduce model complexity (and thus, resource needs). We describe these solutions in turn, and highlight how they all violate the stringent accuracy and/or latency requirements of video analytics.

Space and time sharing. Existing deep learning frameworks recommend allocation at the granularity of an entire GPU, e.g., PyTorch uses a worker-per-model architecture and advocates allocating a GPU per worker [50]. Space-sharing techniques, such as NVIDIA's Multi-Process Service (MPS) [13] and TensorRT inference server [17], eschew this exclusivity and partition the GPU memory per model. Although such space-sharing approaches are effective when the set of models in a workload can fit together in GPU memory, they are insufficient when that property does not hold, which is a common case at the edge (§3.1).

There are two natural solutions for supporting scenarios where multiple models must run on the same GPU, but cannot fit together in that GPU's memory. The first is to simply place models that cannot fit together on *different* GPUs [39, 82]. However, this is a luxury that resource-constrained edge settings cannot afford. The alternative is

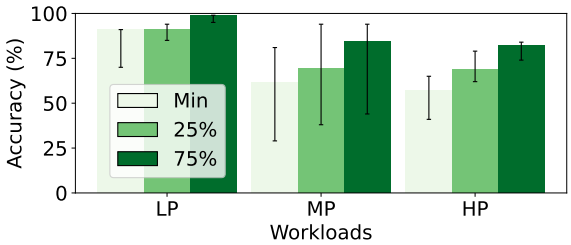

Figure 3: Achieved accuracy with time/space-sharing alone (i.e., using our Nexus variant) for different memory availability (following the definitions in §2). Bars list results for the median workload in each class, with error bars spanning min to max.

to *time share* the model execution in the GPU by *swapping* them in and out of GPU memory [25, 39, 45, 82, 94]. To do this, models are stashed in CPU before they are loaded into the GPU memory through a PCIe interface, which provides bandwidths of up to 64 GB/s. Unfortunately, as we will show next, time-sharing techniques are bottlenecked by the frequent need to swap models, which severely limits their utility. More recently, SwapAdvisor [45] and Antman [94] proposed swapping at finer granularities, e.g., individual or small groups of layers. However, even these approaches are limited in our setting because a handful of layers in vision DNNs typically account for the majority of memory usage (§5.2); edge boxes often lack the GPU memory to concurrently house even these expensive singular layers.

To evaluate the efficacy of time-sharing strategies in handling the edge video analytics memory challenge, we consider a hybrid version that *packs* models into GPU memory, and only executes as many models as possible while ensuring that swapping costs for the next model to run are hidden. More concretely, we extend the Nexus [82] time-sharing system to incorporate such pipelining. Our variant first organizes models in round-robin order (as Nexus does), and profiles the workload offline to determine the best global list of per-model batch sizes that maximizes the minimum achieved per-model throughput while adhering to a pre-specified SLA (i.e., a per-frame processing deadline). Using those batch sizes, the scheduler then traverses the round robin ordering with the goal of minimizing GPU idle time: when loading the next model in the order, if there does not exist sufficient memory to load both parameters and intermediates, the most recently run model (and thus, the one whose next use is in the most distant future) is evicted to make space.

Figure 3 shows the accuracy that the Nexus variant achieves on our workloads with an SLA of 100 ms; we observed similar trends for other common SLAs in video analytics [82]. As shown, accuracy drops are substantial, growing up to 43% relative to a setting when there exists sufficient memory to house all models at once; for context, consider that video analytics pipelines seek accuracy upwards of 99% [44, 63], and new vision DNNs bring accuracy wins of only 2-10% [46,57,85]. Digging deeper, we find that the root cause is the disproportionately high loading times of vision

Model	Load Time (ms)	Run Time (ms)		
		BS=1	BS=2	BS=4
YOLOv3	49.5	17.0	24.0	39.9
ResNet152	73.25	24.81	26.27	26.70
ResNet50	27.1	8.41	8.50	8.52
VGG16	72.2	2.10	2.23	2.40
Tiny YOLOv3	6.7	3.0	3.5	5.2
Faster RCNN	117.3	115.4	210.1	379.4
Inceptionv3	11.8	9.1	9.1	9.1
SSD-VGG	16.1	16.5	25.7	44.6

Table 2: Load/run time for various models and 3 batch sizes.

DNNs that must be incurred when swapping due to memory restrictions. As shown in Table 2, per-model loading delays are 0.98-34.4× larger than the corresponding inference times (for batch size 1). When facing the strict SLAs of video analytics, these loading costs result in the inability to keep pace with incoming frame rates, and thus, dropped (unprocessed) frames; the Nexus variant skipped 19-84% of frames.

Predicting workload characteristics. A natural solution to alleviate memory tensions is to selectively preload models based on predictions of (the importance of) the target workload [99], e.g., deprioritizing inference on certain streams at night due to lack of activity. However, in edge video analytics, spatial correlation between video streams results in model demands being highly correlated [49, 53, 63, 67].

Compression and quantization. These techniques are commonly used to generate lighter model variants that impose lower memory (and more generally, compute)[3] footprints and deliver lower inference times. Some families directly offer off-the-shelf compressed variants (e.g., Tiny YOLOv3), and techniques such as neural architecture search can be used to develop cheaper variants that are amenable to constrained deployment settings [40]. Regardless, in reducing model complexity, these cheaper model variants typically sacrifice accuracy and are more susceptible to drift, relative to their more heavy-weight counterparts [27, 87]; consequently, determining the feasibility of using such models in a given setting requires careful tuning and analysis by domain experts.

We consider compression and quantization as orthogonal to merging for two reasons. First, in common workloads that involve a mix of models and tasks (§2), it may not be possible to compress all of the models while delivering sufficient accuracy. However, even a handful of non-compressed models can exhaust the available GPU memory (§3.1). Second, compressed models exhibit sharing opportunities: our workloads include compressed and non-compressed models (§2), and our results show that GEMEL is effective for both (§6).

[3]There exist training optimizations that trade off memory usage for computation overheads [72, 75, 81]. We eschew such techniques given the holistic constraint on compute resources that edge boxes face (§1).

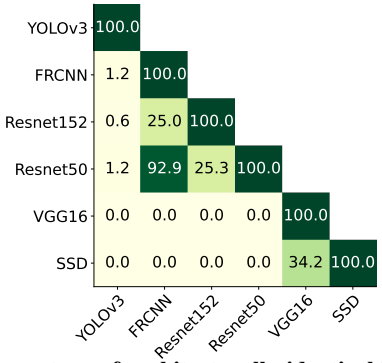

Figure 4: Percentage of architecturally identical layers across different model pairs.

4 Our Approach: Model Merging

To address the high model loading costs that plague existing memory management strategies when workloads cannot fit together in a GPU's memory (§3.2), we propose *model merging*. Model merging is complementary to time/space sharing of GPU memory, and its goal is straightforward: share layers across models in a workload such that only one copy of each shared layer (i.e., the layer definition and weights) must be loaded into GPU memory and can be used during inference for all of the models that include it. The benefits of merging are two-fold: (1) reduce the overall memory footprint of a workload, thereby enabling edge boxes to house more models in parallel and perform fewer swaps (or equivalently, lowering the number of edge boxes needed to run the workload), and (2) accelerate any remaining swaps by reducing the amount of additional memory that the next model to load requires. Note that merging does not involve sharing intermediates (i.e., layer outputs) for a common layer because models may be running on different videos and thus on different input values. We next highlight the promise for merging in edge video analytics (§4.1), and then lay out the challenges associated with realizing merging in practice (§4.2).

4.1 Opportunities

Commonality of layers. To successfully share a layer across a set of models, that layer must be architecturally identical across those models, i.e., the input size, output size, kernel size, stride, etc. must be the same. Figure 4 lists the percentage of layers that are architecturally identical across pairs of representative models in our workloads. The pairs fall into one of three categories: (1) instances of the same model, (2) different models in the same model family (e.g., multiple ResNet models), and (3) different models in different families. Multiple instances of the same model unsurprisingly match on every layer; this favorable scenario is not uncommon in video analytics deployments, as several model architectures tend to dominate the landscape [20] and a given model can be employed on different video feeds or in search of different objects (§2). However, more interestingly, we also observe sharing opportunities across different models

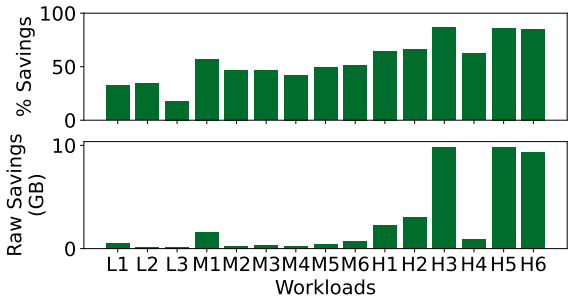

Figure 5: Potential memory savings when all architecturally identical layers are shared across the models in each workload.

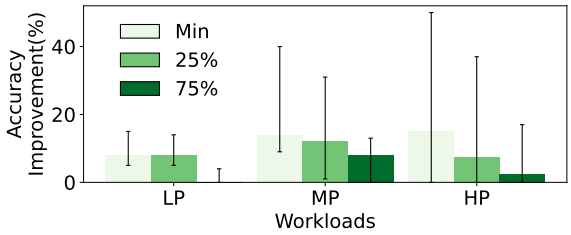

Figure 6: Potential accuracy improvements when sharing all architecturally identical layers. Memory availability is defined in §2, bars list medians, and error bars span min to max.

from the same (up to 25.3%) and divergent families (up to 92.3%). For example, the ResNet50 classifier shares 50 layers with both ResNet152 and the Faster RCNN detector.

Though perhaps surprising, these layer similarities directly follow from the fact that the considered models are all vision processing DNNs. More specifically, they are all ingesting pixel representations of raw images, and employing a series of traditional CV operations [21, 57], e.g., a convolutional layer is applying a learned filter to raw pixel values in preparation for downstream processing. Moreover, the target tasks (e.g., detection, classification) are rooted in identifying and characterizing objects in the scene using low-level CV features such as detected edges and corners [28, 32, 44, 58, 59, 63, 103, 104].

Prior work has capitalized on such similarities for efficient multi-task learning [31, 52, 95] and architecture search [66, 73]. Those efforts aim to reduce computation overheads by sharing "stems" of models, i.e., sharing a group of contiguous layers (and their generated intermediates) starting from the beginning of the models. In contrast, we aim to exploit architectural similarities to reduce memory overheads via merging. As a result, merging only requires layer definitions and weights to be shared, but not generated intermediate values. This distinction is paramount because, as we will discuss in §5.2, memory-heavy layers typically reside towards the end of vision DNNs. Consequently, stem sharing would require almost all model layers to be shared to reap substantial memory savings (i.e., from the beginning up to the memory-heavy layers); doing so typically comes with an unacceptable accuracy drop (§4.2 and §6). Merging (and thus, GEMEL), on the other hand, can share only those memory-heavy layers to simultaneously deliver substantial memory savings and preserve result accuracy.

Potential memory savings and accuracy improvements. We estimated the potential benefits of model merging by sharing all of the common layers across the models in each of our workloads; note that these results represent an *upper bound* on merging benefits as they disregard the challenge of identifying an acceptable set of weights per shared layer (§4.2). As shown in Figure 5, the memory savings are substantial: per-workload memory usage dropped by 17.9-86.4% relative to no merging, translating to raw savings of 0.2-9.9 GB. Importantly, these savings result in 2 and 4 new workloads fitting entirely (i.e., without swapping) on edge boxes with 2 GB and 8 GB of GPU memory (with batch size 1), respectively. Similarly, the number of 2 GB edge boxes needed to support each workload drops from 1-9 to 1-4. We further evaluated the resulting impact on end-to-end accuracy by comparing the performance of the Nexus variant described in §3.2 when run on workloads with and without (maximal) merging. Note that models in both cases were ordered in the same way, to maximize the benefits of merging (we elaborate on this in §5.4). As shown in Figure 6, merging can boost accuracy by up to 50% across our workloads. These benefits are a direct result of lower swapping costs, and the resulting ability to run on 29-61% more frames.

4.2 Challenges

Merging layers for memory reductions requires using shared weights across the models in which those layers appear. However, those shared weights must not result in accuracy violations for any of the models; such accuracy drops would forego the benefits that merging brings from faster swapping. This is a particularly challenging requisite given that the space of models in edge video analytics workloads exhibit heterogeneity in terms of overall architecture, target task, object(s) of interest, and target video(s) (§2). More concretely, there exist two major challenges in practically exploiting the architectural commonalities presented in §4.1.

Challenge 1: sharing vs. accuracy tension. To maximize memory savings (and the corresponding accuracy benefits), merging seeks to share as many architecturally identical layers as possible across a workload's models. However, we observe that accuracy degradations steadily grow as the number of shared layers increases. Figure 7 illustrates this trend by sharing different numbers of identical layers across representative pairs of models that vary on the aforementioned properties, e.g., target task. These results were obtained when we increase the number of shared layers by moving from start to end in the considered models, but similar trends are observed for other selection strategies, e.g., random. The trends also persist for all other model combinations in our workloads.

The reason for this behavior is intuitive: the retraining performed to assess the feasibility of a given sharing configuration is *end-to-end* across the considered models. During this

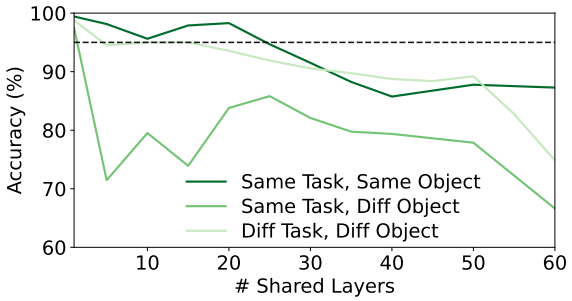

Figure 7: Accuracy after 5 hours of retraining when sharing additional architecturally-identical layers for different model pairs (starting from their origins). Tasks cover detection (Faster RCNN) and classification (ResNet50), and two objects: people, vehicles. Results list the lower per-model accuracies per pair.

process, weights are being tuned for all of the layers in all of the models, with the constraint being that the shared layers each use a unified set of weights. Sharing more layers has three effects: (1) additional constraints are being placed on the training process, (2) it is harder to find weights for (the shrinking number of) unshared layers that simultaneously accommodate the growing number of constraints, and (3) learning each model's desired function becomes more difficult as there exist fewer overall parameters to tune [22, 62]. Indeed, it is for these reasons that isolated merging strategies such as averaging weights across copies of each shared layer (while keeping other layers unchanged) do not suffice; we find that sharing even singular layers in this manner almost always results in unacceptable accuracy violations.

Digging deeper, the issue stems from non-convex optimization of DNNs, which leads to several equally good global minima [55, 56]. Thus, training even two identical models on the same dataset, and for the same task/object, often results in divergent weights across each layer, despite the resultant models exhibiting similar overall functionality.

Challenge 2: retraining costs. The retraining involved in determining whether a given combination of layers to share can meet an accuracy target, and if so, the corresponding weights to use, can be prohibitively expensive. For instance, each epoch when jointly retraining two Faster RCNN models that detect vehicles at nearby intersections (i.e., a simple scenario) consumed ≈35 minutes, and different combinations of layer sharing required between 1-10 epochs to converge. These delays grow as additional models are considered since training data must reflect the behavior of all of the unmerged models that are involved, e.g., by using the original training datasets for each of those models. Worse, it is difficult to know, a priori, which sharing configurations can meet accuracy targets (and which will not) in a reasonable time frame. For instance, as shown in Figure 7, the 'breaking point' for an accuracy target of 95% and training time of 5 hours varies across the model pairs, e.g., from 5-25 layers across the shown model pairs. The result also fails to support the use of intuitive trends to predict the success of different

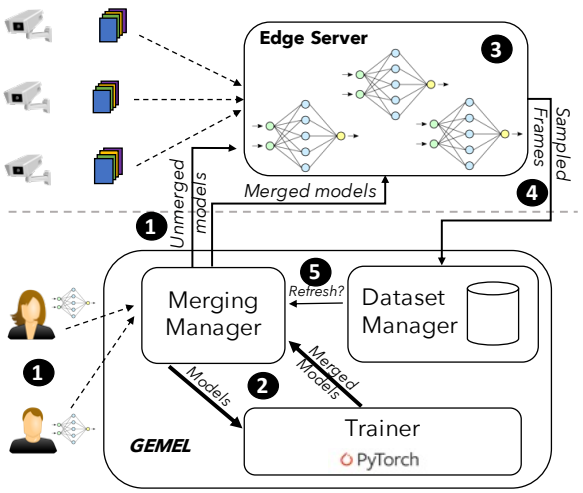

Figure 8: GEMEL **architecture.**

sharing configurations. In particular, models targeting the same task or object do not exhibit any discernible advantage relative to models that vary on those properties.

5 GEMEL Design

GEMEL is an end-to-end system that practically integrates model merging into edge video analytics pipelines by addressing the challenges in §4.2. We first provide an overview of GEMEL's operation, and then describe the core observations (and resulting optimizations) that it leverages to enable timely merging without violating accuracy requirements.

5.1 Overview

Figure 8 illustrates the high-level merging and inference workflows across GEMEL's cloud and edge components. As in existing pipelines [16, 53, 63], users register inference tasks (or "queries") at GEMEL's cloud component by providing a DNN, and specifying the input video feed(s) to run on and the required accuracy for the results.

Upon receiving new queries, GEMEL bootstraps the edge inference process by sending unaltered versions of the registered models to the appropriate edge box(es) ❶. The edge boxes employ existing time/space-sharing schedulers when GPU memory is insufficient to house all of the registered models. By default, GEMEL's edge boxes run the Nexus variant from §3.2 which pipelines inference and model loading to maximize the minimum per-model throughput.

After initiating edge inference for the registered queries, GEMEL's cloud component begins the merging process, during which it *incrementally* searches through the space of potential merging configurations across the registered models, and evaluates the efficacy of each candidate configuration in terms of both its potential memory savings and its ability to meet accuracy requirements ❷. The evaluation of each configuration involves joint retraining and validation of the models participating in merging. Since GEMEL's goal is to ensure that the retrained models deliver sufficient accu-

racy (relative to the original model) on the target video feeds, the data required for these tasks can be obtained in one of two ways: users can explicitly supply the data that was used to train the original models, or GEMEL can automatically generate a training dataset by running the supplied model (or a high-fidelity model performing the same task [53, 100]) on sampled frames from the target feed.

At the end of each merging iteration, if the considered configuration was successfully retrained to meet the specified accuracy targets for all constituent models, GEMEL shares the updated merged models (or more precisely, the updated weights) with the appropriate edge boxes ❸. Newly shared merging results may result in altered edge inference schedules to maximize the benefits that merging delivers in terms of reducing swapping costs and boosting inference throughput. The iterative merging process for the current workload then continues until (1) the cloud resources dedicated to merging have been expended, (2) no configurations that can deliver superior memory savings are left to explore, or (3) models that exhibit sharing opportunities are either newly registered or deleted by users.

To ensure that merged models continually deliver acceptable accuracy (relative to the originals) in the face of dynamic video content, GEMEL periodically assesses *data drift*. To do this, as in prior systems [63, 87], edge servers periodically send sampled frames (and their inference results, if collected) to GEMEL's cloud component ❹. These sampled frames are used to augment the datasets considered for retraining merged models, and to track the accuracy of recent results generated at the edge by deployed merged models. For the latter, GEMEL runs the original, user-provided models on the sampled videos and compares the results to those generated by the merged models. If accuracy is observed to be under the specified target for any query, GEMEL reverts edge inference to use the corresponding original (unmerged) models, and resumes the merging and retraining processes starting with the previously deployed weights ❺.

Implementation. GEMEL uses PyTorch [18] to manage cloud-side retraining (during merging) and edge inference. Overall, GEMEL is implemented in ≈3500 LOC.

5.2 Guiding Observations

Two key empirical observations guide GEMEL's approach to tackling the challenges in §4.2. We describe them in turn, along with the implications on GEMEL's merging strategy.

Observation 1: power-law memory distributions. We find that vision DNNs consistently exhibit power-law distributions in terms of memory usage, whereby a few layers account for most of the overall model's memory consumption. Figure 9 illustrates this, showing that 57-90% of memory usage in representative models is accounted for by fewer than 15% of the layers. Figure 9 also shows that these memory-dominant layers tend to reside towards the end of the corresponding model architectures. For example,

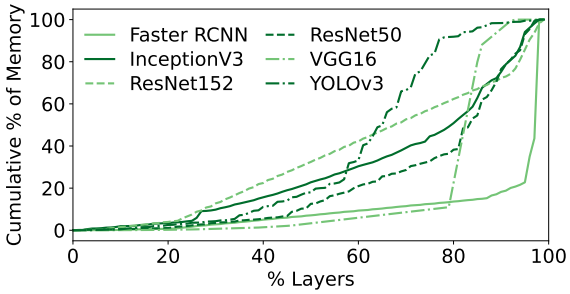

Figure 9: Cumulative memory consumed by each model's layers moving from start to end of the model.

in Faster RCNN, two layers towards the end of the model (in the last 10 layers) account for 85% of the total memory needs; in VGG16, a layer three positions from the end accounts for 73% of the memory consumed by the model; 19% of the layers in YOLOv3 account for 84% of memory usage.

Further investigation of these memory-heavy layers reveals that they are typically of one of two forms. First, the final feature extraction layers in vision DNNs tend to consume more memory than their predecessors because they operate on features that characterize larger regions of the image, i.e., less localized features. Second, task-specific layers that immediately follow feature extraction commonly consume significant memory since they are responsible for pooling the extracted feature values to find, e.g., bounding boxes that house objects; note that these layers precede the (lightweight) final layers that perform the actual predictions.

The implication is that targeting a small set of "heavy-hitter" layers (towards the end of models) for merging can yield most of the potential memory benefits. Consequently, the tension between memory savings and accuracy is far more favorable than that between the number of shared layers and accuracy (from Figure 7).

Observation 2: independence of per-layer merging decisions. In DNNs, layers are configured based on input formats, target task, execution time restrictions, etc., and that ordering may affect either or both of correctness and optimizing execution time. Hence, a natural assumption is that the ability to share any one layer is dependent on sharing decisions for other layers, e.g., a layer may be shareable if and only if other layers are shared. Prior work has highlighted that inter-layer dependencies primarily arise between neighboring layers in a DNN, e.g., with transfer learning, performance drops are largest when splitting neighboring layers, and patterns in accuracy do not translate well with shared layers [96]. Thus, to determine the existence of layer-wise dependencies as it pertains to merging, we focus our analysis on (potential) dependencies between neighboring layers.

For each layer in pairs of identical models in our workloads, we tried multiple forms of sharing: the layer alone, the layer and its immediate neighbors on either side, the layer plus its two nearest neighbors on either side, and so on up to five. In each case, we trained until convergence to deter-

mine whether the layer could be successfully shared or not (accuracy-wise). Across all of our models and accuracy targets, we *never* observed a case in which a layer was unable to meet an accuracy target on its own, but it was able to meet the accuracy target when surrounding layers were also shared. This result is in line with our finding that sharing more layers leads to larger accuracy degradations (Figure 7), and implies that layers can be considered independently during merging without harming their potential merging success.

Takeaway. Collectively, these observations motivate an incremental merging process that attempts to share one new layer at a time, and prioritizes heavy-hitter layers that consume the most memory (and are thus the most fruitful to share). In this manner, memory-heavy layers are considered in the most favorable settings (i.e., with the fewest other shared layers), and each increment only modestly adds to the likelihood of not meeting accuracy targets (as the boundary for what sharing is feasible is discovered). §5.3 details GEMEL's incremental merging heuristic.

Note. Despite arising across our diverse workloads, these observations are not guarantees. Importantly, violation of these observations only results in merging delays (inefficiencies), but not accuracy breaches; accuracy is explicitly vetted prior to shipping merged models to the edge for inference.

5.3 Merging Heuristic

Given a set of registered queries and their models (i.e., a workload), GEMEL begins by enumerating the layers that appear in the workload, and annotating each layer with a listing of which models the layer appears in (and where) and the total memory it consumes across the workload; we refer to all appearances of a given layer as a 'group.' GEMEL then sorts this list in descending order of the amount of memory that each group consumes in the workload, e.g., a 100 MB layer that appears in 4 models would be earlier in the list than a 120 MB layer that appears 3 times. The effect of this sorting is that memory-heavy groups, or those that would result in the largest memory savings if successfully merged, are towards the beginning of the list.

GEMEL then maintains a running merging configuration, and simultaneously merges and trains layers across models in an *incremental* fashion. To begin, GEMEL selects the first group from the sorted list (i.e., the group which consumes the most memory across the workload) and attempts to share it across all of the models in which it appears; this group is added to the running configuration. While a subset of models could be considered instead, GEMEL aggressively opts to first explore sharing across all of the potential models for that group, and then to selectively remove appearances of the layer when the resulting accuracy is insufficient. The reason is that we did not observe any model clustering strategies (e.g., based on task) that identified models consistently unable to share layers without harming accuracy (§4.2).

To retrain and merge the current running configuration, GEMEL selects initial weights for the newly added group from a random model that includes that layer. Retraining then continues until the merged models each meet their respective accuracy targets, or a preset time budget has elapsed (10 epochs by default). If retraining is successful, GEMEL adds the next group in the sorted list to the running configuration, and resumes retraining from the weights at the end of the previous iteration. In addition, the generated merged models are sent to the edge box and immediately incorporated into edge inference (§5.4).

If retraining is not successful at the end of an iteration, GEMEL must decide whether to prune models from the current group and try again, or to discard the group altogether and move on to the next one in the sorted list. To do this, GEMEL follows an additive increase, multiplicative decrease strategy. More specifically, recall that GEMEL first considers a group in its entirety. This behavior is additive in that, each time a new group is considered, the number of shared layers in the merging configuration grows by the size of the group. To counter this, upon unsuccessful retraining, GEMEL halves the current group, eliminating the half of the layer appearances that are closest to the start of the models (since they typically consume less memory). If the resulting layer appearances consume more memory than the next group in the sorted list, GEMEL considers those layers; else, GEMEL entirely removes the current group from the running policy, and moves on to the next group. In either case, retraining resumes from the weights at the end of the last successful iteration. This iterative process continues until there are no additional groups to consider.

Accelerating retraining. Each iteration requires GEMEL to run retraining over many epochs, and validate the results to determine whether the workload's models meet their accuracy targets. In order to accelerate these tasks, GEMEL takes an adaptive approach. During the validation phase, as per-model accuracy values approach their targets, it is often unnecessary to train further on full epochs of data. Instead, GEMEL reduces the amount of training data once the accuracy is within a pre-defined threshold of the target. Specifically, GEMEL reduces the amount of data so that it is inversely proportional to the gap in accuracy normalized by the lift since the previous training. Reducing the data on which the models are trained when detecting such *early success* directly translates to reduced training times. Similarly, GEMEL detects *early failures* by looking at the validation results, and removing models that are not improving at the same pace as the rest of the models over a certain number of epochs (2 by default). We empirically observe that early success and early failure detection drastically (28% on average, in our workloads) reduces the time to train merged models.

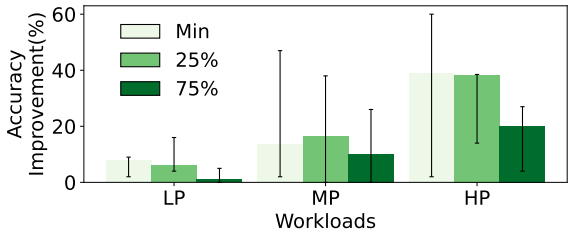

Figure 10: Accuracy improvements with GEMEL compared to time/space-sharing alone for different GPU memories (defined in §2). Bars list median workloads, with error bars as min-max.

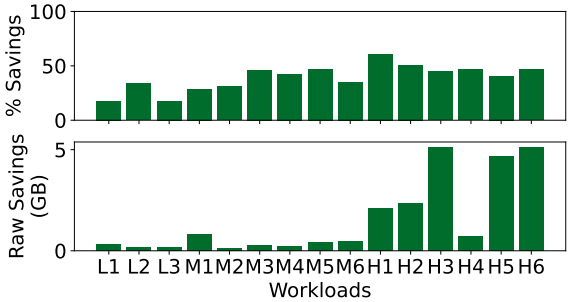

Figure 11: GEMEL's per-workload memory savings.

5.4 Edge Inference

By default, edge inference with GEMEL uses the Nexus variant described in §3.2; however, we note that GEMEL is compatible with any time/space-sharing scheduler. Upon receiving a new set of merged models from GEMEL's cloud component, an edge server directly incorporates those models into its inference schedule. However, to ensure that merging benefits are maximized, the schedule is altered to reduce the amount of data that must be loaded across the anticipated swaps. More specifically, during the offline profiling that Nexus uses to select per-model batch sizes, GEMEL estimates per-workload-iteration swapping delays based on per-model computation costs and swapping delays (which are directly influenced by merging). The key idea is that, when merging is used, in addition to ordering models to reduce the number of swaps, models that share the most layers should be placed next to one another in the round robin order. This lowers the cost of each swap by enabling finer-grained swapping, whereby only those layers in the next model that are not already in GPU memory must be loaded to support inference.

6 Evaluation

Building on the motivational and micro-benchmark results from §3-5, we evaluated GEMEL end-to-end across the diverse workloads and settings from §2. Our key findings are:

- GEMEL improves per-workload accuracies by 8-39% compared to time/space-sharing strategies alone; these improvements result from GEMEL processing 13-44% more frames (while adhering to SLAs).
- GEMEL reduces memory needs by 17.5-60.7% (0.2-5.1 GB), which is 5.9-52.3% larger than Mainstream (state-of-the-art stem sharing), and within 9.3-29.0% of an optimal that ignores weights (and thus accuracy degradations) when sharing layers.
- More than 70% of GEMEL's memory savings are achieved within the first 24-210 minutes of merging+retraining due to its incremental merging heuristic.

6.1 Overall Performance

End-to-end Accuracy Improvements. We first compare GEMEL with time/space-sharing solutions alone, i.e., the Nexus variant from §3.2 that is presented with only unmerged (original) models. Our experiments consider all of the workloads and resource settings from §2, a per-frame processing SLA of 100 ms, and an accuracy target of 95%; trends hold for other accuracy targets and SLAs, and we present results with varying values for each in §6.2.

Figure 10 presents our results, showing that GEMEL improves accuracy by 8.0%, 13.5%, and 39.1% for the median LP, MP, and HP workloads, respectively, when the edge box GPU's memory is just enough to load and run the largest model in each workload, i.e., the *min* setting. The root cause of these benefits is GEMEL's ability to reduce the amount of time blocked on swapping delays by 17.9-84.0%. This, in turn, result in GEMEL being able to process 13-44% more frames than when merging is not used.

Our results highlight two additional points. First, we observe that GEMEL's benefits are highest for workloads that are most significantly bottlenecked by memory restrictions (and thus loading costs). For instance, workloads HP1 and LP1 exhibit largely different memory vs. computation profiles: loading costs are 66% of computation costs in the former, but only 15% in the latter. Accordingly, GEMEL's accuracy improvements across the available memory settings are 11-60% and 5-16% for workloads HP1 and LP1, respectively. Second, Figure 10 also shows that, as expected, the benefits that GEMEL delivers for each workload decreases as the available GPU memory grows, e.g., accuracy improvements drop to 17.5% and 10.2% for the median MP workload when GPU memory grows to 50% and 75% of the total workload memory needs. The reason is straightforward: larger GPU memory yields fewer required swaps even without merging.

Memory Reductions. Figure 11 lists the memory reductions that GEMEL delivers for each considered workload by sharing model layers and the associated weights, i.e., parameter reductions. We note that reported values here are based on GEMEL's final merging results and an accuracy target of 95%; we analyze the incremental nature of GEMEL's merging heuristic in §6.2. As shown, parameter reductions are 17.5-33.9% for LP workloads, 28.6-46.9% for MP workloads, and 40.9-60.7% for HP workloads; the corresponding raw memory savings are 0.2-0.3 GB, 0.2-0.8 GB, and 0.7-5.1 GB, respectively. When analyzed in terms of overall memory usage during inference (i.e., including the parameters, inference framework, and intermediate data generated during

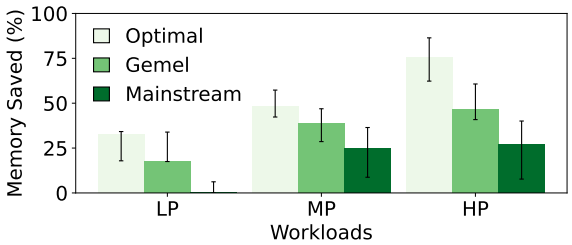

Figure 12: Memory savings with GEMEL, an optimal that ignores accuracy, and Mainstream [52]. Bars list the median workload per class, with error bars spanning min to max.

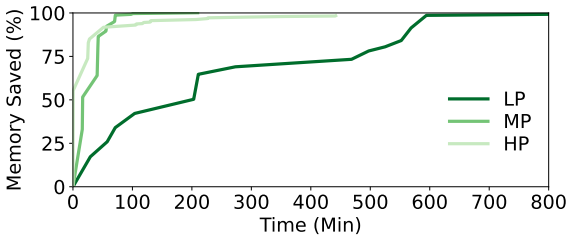

Figure 13: GEMEL's memory savings over time during incremental merging. Results show the median workload per class.

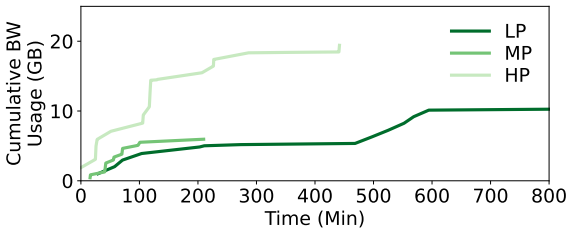

Figure 14: Cloud-edge bandwidth usage during incremental merging. Results show the median workload per class.

model execution), reductions are 4.5-48.1% across the workloads. Wins are generally higher for workloads with larger parameter reductions, with the exception of Workloads LP1 and LP3 (reductions of 6.3% and 4.5%) whose intermediates are particularly large relative to the parameters.

To better contextualize the above memory savings, we compare GEMEL with two alternatives. First, we consider a theoretical optimal (*Optimal*) that shares all layers that are architecturally identical across a workload's models, without considering accuracy (and the need to find shared weights for those layers). Thus, Optimal represents an *upper bound* on GEMEL's potential memory savings. Second, we compare with *Mainstream* [52], a recent stem-sharing approach. To run Mainstream, we trained each model in our workloads several times, each time starting with pre-trained weights (based on ImageNet [78]) and freezing up to different points, e.g., freeze up to layer 10, freeze up to layer 15, etc. We selected the configuration for each model that kept the most layers frozen while meeting the accuracy target (95% relative to no freezing). Then, within each workload, we merged all layers that were shared across the frozen layer set of the constituent models (note that these layers have identical weights) and recorded the resultant memory savings. Note that this evaluation of the merging benefits from stem sharing is optimistic as Mainstream would only be able to share layers across models that operate on the same video streams.

Figure 12 shows our results, from which we draw two conclusions. First, GEMEL's memory savings are within 9.3%, 15.0%, and 29.0% of Optimal for the median LP, MP, and HP workloads, respectively. Second, GEMEL's memory reductions are 5.9-52.3% larger than Mainstream's across all workloads. This is a direct consequence of GEMEL's prioritization of memory-heavy layers that routinely appear towards the end of models (§5.2). By requiring shared stems from the start of the considered models, Mainstream would have to share all layers until it reached the memory-heavy ones; in essence, this amounts to sharing nearly-entire models, which we find is rarely possible while also meeting accuracy targets (Figure 7). In contrast, GEMEL begins by merging only the few memory-heavy layers. We note that the high variance in Mainstream's results are due to the fact that different models drop in accuracy at different rates when more layers are frozen. Classifiers, for example, tend to drop relatively slowly, so classifier-heavy workloads showed more savings (up to 70.1%). In contrast, object detection is a harder task where accuracy drops more quickly; in detection-heavy workloads, Mainstream was unable to share many layers (achieving savings as low as 1.0%).

6.2 Analyzing GEMEL

Incremental memory savings. Key to GEMEL's practicality are its efficient merging heuristic and retraining optimizations that aim to reap memory savings early in the process; indeed, this is important not only to reap accuracy-friendly memory wins quickly, but also to quickly respond to changes in the presented workload, e.g., newly registered models. Figure 13 shows the amount of memory savings that GEMEL unlocks over time during merging. As shown, 73% of GEMEL's achieved memory savings for the median HP workload are realized within the first 24 minutes of merging. Similarly, 86% and 64% of the total memory savings are achieved in the first 42 and 210 minutes of merging for median MP and LP workloads, respectively.

Network bandwidth usage. Recall that, at the end of each successful merging increment, GEMEL ships merging results to the appropriate edge servers for use during inference. More specifically, GEMEL must ship weights for all models that are involved in the considered merging configuration. Consequently, applying each new set of merging results consumes cloud-to-edge bandwidth. Figure 14 parallels the results from Figure 13, but instead reports the amount of bandwidth used over time to ship merging results. As shown, cumulative bandwidth consumption between GEMEL's cloud and edge components during merging is 6.0-19.4 GB for the three workloads. More importantly, bandwidth consumption largely grows after substantial memory savings are already reaped. For example, for the median MP workload, 86% of

	LP	MP	HP
Default	15.3%	12.4%	6.3%
80% accuracy	23.1%	17.3%	10.2%
10 FPS	15.1%	8.0%	5.9%
400 ms SLA	18.2%	14.7%	15.5%

Table 3: GEMEL's accuracy wins (compared to time/space-sharing alone) with varied accuracy targets, FPS, and SLAs. Default uses 95% accuracy target, 30 FPS, and 100 ms SLA. Results use the 'min' memory setting from §2.

memory savings are achieved in 42 minutes, while only 2.1 GB (of the total 6.0 GB) of bandwidth is used during that time. The reason is that later stages of incremental merging explore the larger number of lower-memory layers. Thus, GEMEL is deployable in bandwidth-constrained settings, and could terminate the merging process once a bandwidth limit is hit, often without sacrificing large memory benefits. Note that shipping weights uses cloud-to-edge bandwidth, which is typically far more provisioned in edge settings than edge-to-cloud.

Varying accuracy, SLA, and FPS. Our results thus far have focused on diverse workloads, but fixed accuracy targets, SLAs, and FPS values. To evaluate the impact of each parameter, we conducted a set of experiments using randomly selected models from each class: 1 LP, 1 MP, and 1 HP. In each experiment, we only vary one parameter, while keeping the other two at the fixed values used earlier.

Table 3 presents our results, which highlight three trends. First, GEMEL's accuracy wins over time/space-sharing alone grow (by 3.9-7.8% for the three workloads) as accuracy targets drop (from 95% to 80%). Key to this increase is the fact that GEMEL's memory savings grow with lower accuracy targets because additional layers are able to be merged without accuracy violations, i.e., certain layers failed to meet 95% when merged and trained, but can meet 80%. Second, GEMEL's accuracy benefits drop as input video frame rates (FPS) drop, e.g., from 0.2-4.4% across the workloads when FPS drops from 30 to 10 fps. The reason is that lower FPS values reduce the amount of inference that must be done in any time window (assuming a fixed SLA), which in turn adds tolerance to high loading delays. Third, GEMEL's benefits grow as SLAs become stricter: accuracy wins for the three workloads rise by 2.3-9.2% when SLA drops from 400 ms to 100 ms. This follows from the fact that tighter SLAs are more sensitive to swapping delays, i.e., each swapping delay amounts to more frames whose processing must be skipped.

7 Additional Related Work

Model Sharing. Sharing parts of models has been explored in the ML literature [79], and more recently in video analytics through Mainstream [52], which aims to share outputs from common *stems* of early layers across models. However, unlike GEMEL which addresses memory bottlenecks, the main goal of these works is to reduce computation. This leads to two limitations for our problem. First, these approaches only apply to models that operate on the same underlying data, limiting their applicability to realistic workloads with many videos. Second, and more importantly, because memory-heavy layers are often towards the end of vision DNNs (§5.2), stem-sharing approaches would have to share almost all model layers to reap large memory savings – doing so almost always comes with accuracy violations (§4.2 and §6.1). In contrast, by only sharing weights (not intermediates), GEMEL is able to quickly share only late-stage layers that enable memory savings and accuracy preservation.

PRETZEL [60] focuses on reusing operators in *non-deep* ML models, such as featurizers. The key observation PRETZEL makes is that *both the operator and the parameters are shared* across models, and hence storing them in a shared object store for reuse leads to better memory utilization; the vast majority of savings in PRETZEL arise from sharing parameters. To extend PRETZEL to DNNs (our focus), the assumption that needs to hold is that shared layers across models have the same weights. However, in edge video analytics, this assumption is often violated as it is common for vision DNNs to be specialized to diverse tasks, objects, and videos (§4.2). Indeed, the core challenges that GEMEL tackles are in (quickly) determining which architecturally identical layers consume substantial memory *and* can be retrained to use unified weights without violating accuracy requirements.

Multi-task learning is a well-known technique in machine learning that can learn multiple *related* tasks simultaneously [31]. Some works also study how to share layers in multi-task learning [86,88]. However, the problem is usually studied in the context of transfer learning, where one model does not have enough data to train on, and instead is trained together with another model; this is in contrast to our setting which considers two sets of pretrained weights that must be shared. Additionally, the related tasks are usually variations of the same model (e.g., spam classification) and thus the data for each individual task can be pooled together. In contrast, objectives vary across edge video analytics DNNs, e.g., detection vs. classification, different objects.

Optimized Video Analytics Pipelines. Chameleon [53] profiles pipeline knobs to identify computationally cheaper configurations that preserve accuracy. VideoStorm [100] proposes scheduling techniques that leverage lag tolerance to optimize analytics results [44] and NoScope [54] build support for low-latency queries on large scale video streams. MCDNN [42] and DeepDecision [76] consider edge resources as part of an optimization problem to jointly run models between and across edge and cloud servers. Finally, multiple systems filter or re-encode frames to reduce both computation and bandwidth costs in video analytics pipelines [30,32,35,49,63,93,101]. These systems are com-

plementary to GEMEL, which focuses on alleviating memory (not compute) bottlenecks in edge video analytics.

8 Conclusion

Model merging is a new memory management technique that exploits architectural similarities across vision DNNs by sharing their common layers (including parameters but not intermediates). Our end-to-end system, GEMEL, efficiently carries out model merging on user-registered DNNs by leveraging several key observations to quickly find and retrain accuracy-preserving layer sharing configurations, and schedule them to maximize merging benefits during edge inference. Experiments on a variety of workloads show that GEMEL reduces memory usage by up to 60.7%, and boosts accuracy by 8-39% compared to time/space-sharing alone.

References

[1] Absolutely everywhere in beijing is now covered by police video surveillance. https://qz.com/518874/.

[2] Are we ready for ai-powered security cameras? https://thenewstack.io/are-we-ready-for-ai-powered-security-cameras/.

[3] "aws outposts". https://aws.amazon.com/outposts/.

[4] "azure stack edge". https://azure.microsoft.com/en-us/products/azure-stack/edge/.

[5] British transport police: Cctv. http://www.btp.police.uk/advice_and_information/safety_on_and_near_the_railway/cctv.aspx.

[6] Can 30,000 cameras help solve chicago's crime problem? https://www.nytimes.com/2018/05/26/us/chicago-police-surveillance.html.

[7] Edge computing at chick-fil-a. https://medium.com/@cfatechblog/edge-computing-at-chick-fil-a-7d67242675e2.

[8] "nvidia jetson: The ai platform for edge computing". https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/.

[9] Paris hospitals to get 1,500 cctv cameras to combat violence against staff. https://bit.ly/2OYiBz2.

[10] Powering the edge with ai in an iot world. https://www.forbes.com/sites/forbestechcouncil/2020/04/06/powering-the-edge-with-ai-in-an-iot-world/.

[11] Video analytics applications in retail - beyond security. https://www.securityinformed.com/insights/co-2603-ga-co-2214-ga-co-1880-ga.16620.html/.

[12] The vision zero initiative. http://www.visionzeroinitiative.com/.

[13] Cuda multi-process service, April 2021.

[14] Edge AI Box. https://hailo.ai/reference-platform/edge-ai-box/, 2021.

[15] Live Video Analytics with Microsoft Rocket for reducing edge compute costs, May 2021.

[16] Microsoft rocket video analytics platform, April 2021.

[17] Nvidia tensorrt, April 2021.

[18] Pytorch, April 2021.

[19] Pytorch-yolov3. https://github.com/eriklindernoren/PyTorch-YOLOv3, 2021.

[20] Traffic Video Analytics – Case Study Report, May 2021.

[21] M. Alam, M. Samad, L. Vidyaratne, A. Glandon, and K. Iftekharuddin. Survey on deep neural networks in speech and vision systems. *Neurocomputing*, 417:302–321, 2020.

[22] Z. Allen-Zhu, Y. Li, and Y. Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. *CoRR*, abs/1811.04918, 2018.

[23] Amazon. Rekognition. https://aws.amazon.com/rekognition/.

[24] G. Ananthanarayanan, V. Bahl, L. Cox, A. Crown, S. Nogbahi, and Y. Shu. Video analytics - killer app for edge computing. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '19, pages 695–696, New York, NY, USA, 2019. Association for Computing Machinery.

[25] Z. Bai, Z. Zhang, Y. Zhu, and X. Jin. PipeSwitch: Fast pipelined context switching for deep learning applications. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 499–514, 2020.

[26] Z. Bai, Z. Zhang, Y. Zhu, and X. Jin. Pipeswitch: Fast pipelined context switching for deep learning applications. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 499–514. USENIX Association, Nov. 2020.

[27] R. Bhardwaj, Z. Xia, G. Ananthanarayanan, J. Jiang, N. Karianakis, Y. Shu, K. Hsieh, V. Bahl, and I. Stoica. Ekya: Continuous learning of video analytics models on edge compute servers, 2020.

[28] S. Brutzer, B. Hoferlin, and G. Heidemann. Evaluation of background subtraction techniques for video surveillance. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 1937–1944, Washington, DC, USA, 2011. IEEE Computer Society.

[29] Z. Cai, M. Saberian, and N. Vasconcelos. Learning complexity-aware cascades for deep pedestrian detection. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 3361–3369, Washington, DC, USA, 2015. IEEE Computer Society.

[30] C. Canel, T. Kim, G. Zhou, C. Li, H. Lim, D. G. Andersen, M. Kaminsky, and S. R. Dulloor. Scaling video analytics on constrained edge nodes. In *2nd SysML Conference*, 2019.

[31] R. Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[32] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 155–168, 2015.

[33] M. Chow, D. Meisner, J. Flinn, D. Peek, and T. F. Wenisch. The mystery machine: End-to-end performance analysis of large-scale internet services. OSDI, 2014.

[34] S. R. E. Datondji, Y. Dupuis, P. Subirats, and P. Vasseur. A survey of vision-based traffic monitoring of road intersections. *Trans. Intell. Transport. Sys.*, 17(10):2681–2698, Oct. 2016.

[35] K. Du, A. Pervaiz, X. Yuan, A. Chowdhery, Q. Zhang, H. Hoffmann, and J. Jiang. Server-driven video streaming for deep learning inference. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 557–570, New York, NY, USA, 2020. Association for Computing Machinery.

[36] M. Everingham, L. Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *Int. J. Comput. Vision*, 88(2):303–338, June 2010.

[37] Google. Google edge network. https://peering.google.com/#/infrastructure, 2016.

[38] Google. Cloud vision api. https://cloud.google.com/vision, 2021.

[39] A. Gujarati, R. Karimi, S. Alzayat, W. Hao, A. Kaufmann, Y. Vigfusson, and J. Mace. Serving dnns like clockwork: Performance predictability from the bottom up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 443–462. USENIX Association, Nov. 2020.

[40] P. Guo, B. Hu, and W. Hu. Mistify: Automating DNN model porting for on-device inference at the edge. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 705–719. USENIX Association, Apr. 2021.

[41] B. Han, F. Qian, L. Ji, and V. Gopalakrishnan. Mp-dash: Adaptive video streaming over preference-aware multipath. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '16, pages 129–143, New York, NY, USA, 2016. ACM.

[42] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, page 123–136, New York, NY, USA, 2016. Association for Computing Machinery.

[43] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[44] K. Hsieh, G. Ananthanarayanan, P. Bodik, S. Venkataraman, P. Bahl, M. Philipose, P. B. Gibbons, and O. Mutlu. Focus: Querying large video datasets with low latency and low cost. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 269–286, Carlsbad, CA, Oct. 2018. USENIX Association.

[45] C.-C. Huang, G. Jin, and J. Li. Swapadvisor: Pushing deep learning beyond the gpu memory limit via smart swapping. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 1341–1355, New York, NY, USA, 2020. Association for Computing Machinery.

[46] J. Hui. Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3). https://jonathan-hui.medium.com/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359, 2018.

[47] C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose. Videoedge: Processing camera streams using hierarchical clusters. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 115–131, Oct 2018.

[48] IBM. Maximo remote monitoring. https://www.ibm.com/products/maximo/remote-monitoring, 2021.

[49] S. Jain, X. Zhang, Y. Zhou, G. Ananthanarayanan, J. Jiang, Y. Shu, V. Bahl, and J. Gonzalez. Spatula: Efficient cross-camera video analytics on large camera networks. In *ACM/IEEE Symposium on Edge Computing (SEC 2020)*, November 2020.

[50] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang. Analysis of large-scale multi-tenant GPU clusters for DNN training workloads. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 947–960, Renton, WA, July 2019. USENIX Association.

[51] M. Jeon, S. Venkataraman, J. Qian, A. Phanishayee, W. Xiao, and F. Yang. Multi-tenant gpu clusters for deep learning workloads: Analysis and implications. *Technical report, Microsoft Research*, 2018.

[52] A. H. Jiang, D. L.-K. Wong, C. Canel, L. Tang, I. Misra, M. Kaminsky, M. A. Kozuch, P. Pillai, D. G. Andersen, and G. R. Ganger. Mainstream: Dynamic stem-sharing for multi-tenant video processing. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 29–42, Boston, MA, July 2018. USENIX Association.

[53] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica. Chameleon: Scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 253–266, New York, NY, USA, 2018. Association for Computing Machinery.

[54] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia. Noscope: Optimizing neural network queries over video at scale. *Proc. VLDB Endow.*, 10(11):1586–1597, Aug. 2017.

[55] K. Kawaguchi, J. Huang, and L. P. Kaelbling. Every local minimum value is the global minimum value of induced model in nonconvex machine learning. *Neural Computation*, 31(12):2293–2323, Dec 2019.

[56] K. Kawaguchi and L. P. Kaelbling. Elimination of all bad local minima in deep learning. *CoRR*, abs/1901.00279, 2019.

[57] S. H. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah. Transformers in vision: A survey. *CoRR*, abs/2101.01169, 2021.

[58] H. Kim, S. Leutenegger, and A. J. Davison. Real-time 3D reconstruction and 6-DoF tracking with an event camera. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VI*, pages 349–364, 2016.

[59] B. Kueng, E. Mueggler, G. Gallego, and D. Scaramuzza. Low-latency visual odometry using event-based feature tracks. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 16–23, Oct 2016.

[60] Y. Lee, A. Scolari, B.-G. Chun, M. D. Santambrogio, M. Weimer, and M. Interlandi. PRETZEL: Opening the black box of machine learning prediction serving systems. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 611–626, Carlsbad, CA, Oct. 2018. USENIX Association.

[61] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua. A convolutional neural network cascade for face detection. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5325–5334, June 2015.

[62] Y. Li and Y. Liang. Learning overparameterized neural networks via stochastic gradient descent on structured data. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 8168–8177, Red Hook, NY, USA, 2018. Curran Associates Inc.

[63] Y. Li, A. Padmanabhan, P. Zhao, Y. Wang, G. H. Xu, and R. Netravali. Reducto: On-camera filtering for resource-efficient real-time video analytics. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 359–376, New York, NY, USA, 2020. Association for Computing Machinery.

[64] Z. Li, Y. Shu, G. Ananthanarayanan, L. Shangguan, K. Jamieson, and V. Bahl. Spider: A multi-hop millimeter-wave network for live video analytics. In *ACM/IEEE Symposium on Edge Computing*. ACM/IEEE, December 2021.

[65] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, July 2017.

[66] H. Liu, K. Simonyan, and Y. Yang. DARTS: differentiable architecture search. *CoRR*, abs/1806.09055, 2018.

[67] X. Liu, P. Ghosh, O. Ulutan, B. S. Manjunath, K. Chan, and R. Govindan. Caesar: Cross-camera complex activity recognition. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, SenSys '19, page 232–244. Association for Computing Machinery, 2019.

[68] Microsoft. Enabling Data Residency and Data Protection in Microsoft Azure Regions. https://azure.microsoft.com/en-us/resources/achieving-compliant-data-residency-and-security-with-azure/, 2021.

[69] S. A. Noghabi, L. Cox, S. Agarwal, and G. Ananthanarayanan. The emerging landscape of edge computing. *GetMobile: Mobile Comp. and Comm.*, 23(4):11–20, May 2020.

[70] E. Nygren, R. K. Sitaraman, and J. Sun. The akamai network: A platform for high-performance internet applications. *SIGOPS*, 2010.

[71] OfCom. Residential landline and fixed broadband services. https://www.ofcom.org.uk/__data/assets/pdf_file/0015/113640/landline-broadband.pdf, 2017.

[72] X. Peng, X. Shi, H. Dai, H. Jin, W. Ma, Q. Xiong, F. Yang, and X. Qian. Capuchin: Tensor-based gpu memory management for deep learning. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 891–905. Association for Computing Machinery, 2020.

[73] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. *CoRR*, abs/1802.03268, 2018.

[74] R. Poddar, G. Ananthanarayanan, S. Setty, S. Volos, and R. A. Popa. Visor: Privacy-preserving video analytics as a cloud service. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1039–1056. USENIX Association, Aug. 2020.

[75] S. Rajbhandari, O. Ruwase, J. Rasley, S. Smith, and Y. He. Zero-infinity: Breaking the GPU memory wall for extreme scale deep learning. *CoRR*, abs/2104.07857, 2021.

[76] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen. Deepdecision: A mobile deep learning framework for edge video analytics. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 1421–1429, 2018.

[77] H. Rebecq, T. Horstschaefer, and D. Scaramuzza. Real-time visual-inertial odometry for event cameras using keyframe-based nonlinear optimization. In *British Machine Vision Conference 2017, BMVC 2017, London, UK, September 4-7, 2017*, 2017.

[78] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[79] S. S. Sarwar, A. Ankit, and K. Roy. Incremental learning in deep convolutional neural networks using partial network sharing. *IEEE Access*, 8:4615–4628, 2019.

[80] J. Sevilla, P. Villalobos, and J. Cerón. Parameter counts in Machine Learning. https://www.lesswrong.com/posts/GzoWcYibWYwJva8aL/parameter-counts-in-machine-learning, 2021.

[81] A. Shah, C. Wu, J. Mohan, V. Chidambaram, and P. Krähenbühl. Memory optimization for deep networks. *CoRR*, abs/2010.14501, 2020.

[82] H. Shen, L. Chen, Y. Jin, L. Zhao, B. Kong, M. Philipose, A. Krishnamurthy, and R. Sundaram. Nexus: A gpu cluster engine for accelerating dnn-based video analysis. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, pages 322–337, New York, NY, USA, 2019. Association for Computing Machinery.

[83] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.

[84] Sony. REA-C1000 Edge Analytics Appliance. https://pro.sony/ue_US/products/ptz-cameras/rea-c1000-edge-analytics-appliance, 2021.

[85] F. Sultana, A. Sufian, and P. Dutta. Evolution of image segmentation using deep convolutional neural network: A survey. *Knowledge-Based Systems*, 201-202:106062, 2020.

[86] X. Sun, R. Panda, R. Feris, and K. Saenko. Adashare: Learning what to share for efficient deep multi-task learning. *arXiv preprint arXiv:1911.12423*, 2019.

[87] A. Suprem, J. Arulraj, C. Pu, and J. Ferreira. Odin: Automated drift detection and recovery in video analytics. *Proc. VLDB Endow.*, 13(12):2453–2465, July 2020.

[88] S. Vandenhende, S. Georgoulis, B. De Brabandere, and L. Van Gool. Branched multi-task networks: deciding what layers to share. *arXiv preprint arXiv:1904.02920*, 2019.

[89] L. M. Vaquero and L. Rodero-Merino. Finding your way in the fog: Towards a comprehensive definition of fog computing. *CCR*, 44(5):27–32, Oct. 2014.

[90] A. R. Vidal, H. Rebecq, T. Horstschaefer, and D. Scaramuzza. Ultimate slam? combining events, images, and IMU for robust visual SLAM in HDR and high-speed scenarios. *IEEE Robotics and Automation Letters*, 3(2):994–1001, 2018.

[91] J. Wang, Z. Feng, S. George, R. Iyengar, P. Pillai, and M. Satyanarayanan. Towards scalable edge-native applications. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, SEC '19, page 152–165, New York, NY, USA, 2019. Association for Computing Machinery.

[92] M. Wang, C. Meng, G. Long, C. Wu, J. Yang, W. Lin, and Y. Jia. Characterizing deep learning training workloads on alibaba-pai, 2019.

[93] Y. Wang, W. Wang, J. Zhang, J. Jiang, and K. Chen. Bridging the edge-cloud barrier for real-time advanced vision analytics. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, Renton, WA, July 2019. USENIX Association.

[94] W. Xiao, S. Ren, Y. Li, Y. Zhang, P. Hou, Z. Li, Y. Feng, W. Lin, and Y. Jia. Antman: Dynamic scaling on GPU clusters for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 533–548, 2020.

[95] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3320–3328, Cambridge, MA, USA, 2014. MIT Press.

[96] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3320–3328, Cambridge, MA, USA, 2014. MIT Press.

[97] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98:289–330, 2019.

[98] A. R. Zamani, M. Zou, J. Diaz-Montes, I. Petri, O. Rana, A. Anjum, and M. Parashar. Deadline constrained video analysis via in-transit computational environments. *IEEE Transactions on Services Computing*, 13(1):59–72, 2020.

[99] X. Zeng, B. Fang, H. Shen, and M. Zhang. Distream: Scaling live video analytics with workload-adaptive distributed edge intelligence. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, SenSys '20, page 409–421, New York, NY, USA, 2020. Association for Computing Machinery.

[100] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman. Live video analytics at scale with approximation and delay-tolerance. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 377–392, Boston, MA, Mar. 2017. USENIX Association.

[101] T. Zhang, A. Chowdhery, P. Bahl, K. Jamieson, and S. Banerjee. The design and implementation of a wireless video surveillance system. pages 426–438, 09 2015.

[102] T. Zhang, A. Chowdhery, P. V. Bahl, K. Jamieson, and S. Banerjee. 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, pages 426–438, New York, NY, USA, 2015. ACM.

[103] A. Z. Zhu, N. Atanasov, and K. Daniilidis. Event-based feature tracking with probabilistic data association. In *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*, pages 4465–4470, 2017.

[104] A. Z. Zhu, N. Atanasov, and K. Daniilidis. Event-based visual inertial odometry. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5816–5824, July 2017.

A Appendix: Workload Details

Model	Video Feed	Object
frcnn-r101	A1	people
r101	A1	person, car, bus, truck
r50	A2	person, car, bus, truck
r152	A3	person, vehicle
mnet	A4	person, car, truck
yolo	A5	people
tiny-yolo	A1	people
ssd-vgg	A6	cars
ssd-vgg	A1	cars
ssd-mnet	A5	cars
ssd-mnet	A4	cars
ssd-mnet	A6	cars
inception	A3	person, vehicle

Table 4: Workload LP1

Model	Video Feed	Object
r152	B1	person, vehicle
r101	B2	person, car, bus, truck
ssd-vgg	B3	people

Table 5: Workload LP2

Model	Video Feed	Object
ssd-mnet	B4	cars
frcnn-r101	B3	people
r152	B1	person, vehicle
r18	B3	person, car, bus, truck, motorbike
inception	B1	person, vehicle

Table 6: Workload LP3

Model	Video Feed	Object
frcnn-r50	B1	cars
frcnn-r50	B1	people
r50	B2	person, car, bus, truck
r50	B1	person, vehicle
r152	B3	person, car, bus, truck, motorbike
r152	B4	person, car, bus, truck
r18	B5	person, car, bus, truck
r18	B4	person, car, bus, truck
tiny-yolo	B3	cars
tiny-yolo	B2	cars
yolo	B5	cars
yolo	B1	cars
ssd-vgg	B4	cars
ssd-vgg	B3	people
inception	B3	person, car, bus, truck, motorbike

Table 7: Workload MP1

Model	Video Feed	Object
r50	B3	person, car, bus, truck, motorbike
r50	B1	person, vehicle
r152	B3	person, car, bus, truck, motorbike
r18	B5	person, car, bus, truck
ssd-mnet	B1	cars
ssd-mnet	B2	cars

Table 8: Workload MP2

Model	Video Feed	Object
frcnn-r101	B4	cars
frcnn-r101	B5	cars
frcnn-r101	B1	cars
frcnn-r101	B2	cars
frcnn-r50	B1	people
r50	B3	person, car, bus, truck, motorbike
r18	B3	person, car, bus, truck, motorbike
ssd-mnet	B3	people
ssd-mnet	B1	people
mnet	B4	person, car, bus, truck
yolo	B3	people
tiny-yolo	B5	cars
tiny-yolo	B1	people
vgg	B4	person, car, bus, truck
inception	B2	person, car, bus, truck
inception	B3	person, car, bus, truck, motorbike

Table 14: Workload HP2

Model	Video Feed	Object
frcnn-r50	A3	cars
frcnn-r50	A3	people
frcnn-r50	A1	cars
frcnn-r50	A1	people
frcnn-r50	A5	cars
frcnn-r50	A5	people
frcnn-r50	A2	cars
frcnn-r50	A4	cars
frcnn-r50	A2	trucks
frcnn-r101	A3	people
yolo	A3	cars
yolo	A3	people
yolo	A1	people
yolo	A7	buses
yolo	A7	cars
yolo	A7	people
yolo	A7	trucks
yolo	A5	trucks
yolo	A5	people
yolo	A6	cars
r152	A3	person, vehicle
r152	A1	person, car, bus, truck
r152	A7	person, car, bus, truck
r152	A6	car, bus, truck
r152	A2	person, car, bus, truck
r152	A4	person, car, truck
r50	A3	person, vehicle
r50	A7	person, car, bus, truck
r50	A6	car, bus, truck
r50	A2	person, car, bus, truck
r50	A6	car, bus, truck
ssd-vgg	A3	people
ssd-vgg	A1	cars
ssd-vgg	A5	people
ssd-vgg	A6	cars
ssd-vgg	A4	cars
vgg	A2	person, car, bus, truck
r18	A2	person, car, bus, truck

Table 15: Workload HP3

Model	Video Feed	Object
yolo	B1	cars
yolo	B5	cars
tiny-yolo	B2	cars
tiny-yolo	B1	cars
tiny-yolo	B3	people
ssd-vgg	B5	cars
ssd-vgg	B3	people
ssd-mnet	B5	cars
ssd-mnet	B3	people
ssd-mnet	B2	cars
ssd-mnet	B1	people
mnet	B3	person, car, bus, truck, motorbike
mnet	B5	person, car, bus, truck
r152	B4	person, car, bus, truck
r152	B3	person, car, bus, truck, motorbike
r152	B1	person, vehicle

Table 16: Workload HP4

Model	Video Feed	Object
frcnn-r50	A3	cars
frcnn-r50	A3	people
frcnn-r50	A1	cars
frcnn-r50	A1	people
frcnn-r50	A5	cars
frcnn-r50	A5	people
frcnn-r50	A2	cars
frcnn-r50	A4	cars
frcnn-r50	A2	trucks
frcnn-r101	A3	people
yolo	A3	cars
yolo	A3	people
yolo	A1	people
yolo	A7	buses
yolo	A7	cars
yolo	A7	people

Table 18: Workload HP6

Model	Video Feed	Object
r101	A1	person, car, bus, truck
r101	A7	person, car, bus, truck
r101	A6	car, bus, truck
r101	A1	person, car, bus, truck
r152	A3	person, vehicle
r152	A1	person, car, bus, truck
r152	A7	person, car, bus, truck
r152	A6	car, bus, truck
r152	A2	person, car, bus, truck
r152	A4	person, car, truck
r50	A3	person, vehicle
r50	A7	person, car, bus, truck
r50	A6	car, bus, truck
r50	A2	person, car, bus, truck
r50	A6	car, bus, truck
tiny-yolo	A1	people
tiny-yolo	A5	people
inception	A3	person, vehicle
inception	A1	person, car, bus, truck
inception	A7	person, car, bus, truck
inception	A6	car, bus, truck
inception	A4	person, car, truck
vgg	A2	person, car, bus, truck
r18	A2	person, car, bus, truck
r18	A2	person, car, bus, truck
r18	A2	person, car, bus, truck

Table 19: Workload HP6 (continued)