OneAdapt: Fast Adaptation for Deep Learning Applications via Backpropagation

Kuntai Du, Yuhan Liu, Yitian Hao, Qizheng Zhang[‡],
Haodong Wang, Yuyang Huang, Ganesh Ananthanarayanan[†], Junchen Jiang
University of Chicago [‡]*Stanford University* [†]*Microsoft Research*

ABSTRACT

Deep learning inference on streaming media data, such as object detection in video or LiDAR feeds and text extraction from audio waves, is now ubiquitous. To achieve high inference accuracy, these applications typically require significant network bandwidth to gather high-fidelity data and extensive GPU resources to run deep neural networks (DNNs). While the high demand for network bandwidth and GPU resources *could* be substantially reduced by *optimally adapting* the configuration knobs, such as video resolution and frame rate, current adaptation techniques fail to meet three requirements simultaneously: adapt configurations *(i)* with minimum extra GPU or bandwidth overhead *(ii)* to reach near-optimal decisions based on how the data affects the final DNN's accuracy, and *(iii)* do so for a range of configuration knobs. This paper presents OneAdapt, which meets these requirements by leveraging a gradient-ascent strategy to adapt configuration knobs. The key idea is to embrace DNNs' *differentiability* to quickly estimate the accuracy's gradient to each configuration knob, called AccGrad. Specifically, OneAdapt estimates AccGrad by multiplying two gradients: InputGrad (*i.e.,* how each configuration knob affects the input to the DNN) and DNNGrad (*i.e.,* how the DNN input affects the DNN inference output). We evaluate OneAdapt across five types of configurations, four analytic tasks, and five types of input data. Compared to state-of-the-art adaptation schemes, OneAdapt cuts bandwidth usage and GPU usage by 15-59% while maintaining comparable accuracy or improves accuracy by 1-5% while using equal or fewer resources.

SoCC '23, October 30–November 1, 2023, Santa Cruz, CA, USA

ACM ISBN 979-8-4007-0387-4/23/10…$15.00
https://doi.org/10.1145/3620678.3624653

CCS CONCEPTS

• **Information systems** → **Data analytics**; • **Computer systems organization** → **Client-server architectures**; • **Computing methodologies** → **Computer vision problems**; • **Theory of computation** → *Discrete optimization.*

KEYWORDS

data analytics, configuration adaptation, backpropagation

ACM Reference Format:
Kuntai Du, Yuhan Liu, Yitian Hao, Qizheng Zhang, Haodong Wang, Yuyang Huang, Ganesh Ananthanarayanan, Junchen Jiang. 2023. OneAdapt: Fast Adaptation for Deep Learning Applications via Backpropagation. In *ACM Symposium on Cloud Computing (SoCC '23), October 30–November 1, 2023, Santa Cruz, CA, USA.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3620678.3624653

1 INTRODUCTION

Many real-world applications run deep neural networks (DNNs) to perform analytics on streaming media data, such as RGB videos, LiDAR point clouds, depth videos, and audio waves. For example, autonomous-driving applications rely on DNNs to detect vehicles in individual video RGB frames and LiDAR point clouds [1, 13, 14, 16, 24, 50, 59, 69, 78–80]. Similarly, smart-home applications use DNNs to extract text from audio segments [6, 20, 23]. We focus on these applications, which we refer to as ***streaming media analytics*** (§2.1). Notably, streaming media analytics encompass the popular video analytics applications, but not all DNN-based tasks (*e.g.,* generative tasks).

Many streaming media analytics systems can be resource-intensive, in ***network bandwidth*** or ***GPU cycles*** or both. To achieve high accuracy, they run complex DNN inference on each frame (or segment) and require data at high fidelity, which potentially requires high GPU usage and network bandwidth (if a remote sensor captures the data).

To reduce resource usage, many prior solutions (*e.g.,* [32, 42, 43, 66, 67, 92, 94, 95]) apply *input filtering* to downsample or drop redundant data and then run *DNN inference* to analyze the filtered input. Ideally, if the input filtering is configured with an optimal setting for its *knobs* (*e.g.,* video frame rate and resolution), the bandwidth usage and GPU usage can be drastically reduced without affecting inference accuracy. (Table 2 summarizes some popular filtering knobs.)

The challenge of optimally adapting the filtering knobs is that the optimal setting of these knobs ***varies*** over time as input data evolves [61, 66, 88, 89, 91, 92]. We call the optimal setting of these knobs an *optimal configuration*. For example, when vehicles stop at the red light, feeding the traffic video at a low frame rate to a DNN can still accurately detect vehicles, but when vehicles start moving fast, the DNN must run more frequently (*e.g.,* 30fps) to detect the vehicles. In this case, the frame rate must be adapted over time.

To handle this challenge, an ideal adaptation logic should meet three requirements:

- *(i) Frequent:* The adaptation logic can run frequently with minimum GPU computation and bandwidth overhead.
- *(ii) Near-optimal:* The adaptation logic can pick a near-optimal configuration based on how various filtering on the data will affect the output of a particular final DNN.
- *(iii) Generic:* The same logic can be applied to a wide range of analytic tasks, streaming media, and filtering knobs.

Prior approaches: Prior efforts fall short in at least one of these requirements (detailed discussion in §2.3).

- *Profiling-based* methods (*e.g.,* [61, 88, 89, 91]) periodically run *extra* DNN inferences to profile the accuracy of alternative configurations, which incur significant GPU overhead. For example, AWStream [89] profiles different combinations of resolutions, frame rates, and quantization parameters. Even after *downsampling* the combinations, such profiling still has up to 17× more GPU computation than regular DNN inference. With limited GPU resources, this profiling overhead decelerates the adaptation of configuration, leading to outdated configuration when the content of input data varies over time (as shown in Figure 1).
- Alternatively, many *heuristic-based* methods (*e.g.,* [32, 42, 43, 66, 67, 81, 92, 94, 95]) avoid extra inference and instead adapt configurations more frequently using cheap heuristics. These heuristics make simplified assumptions about which parts of the input are unimportant to the final DNN, resulting in low accuracy (as shown in Figure1). For instance, some heuristics filter a new frame for DNN inference when its pixels differ from the last analyzed frame significantly (*e.g.,* [11, 32, 66]), but the pixel differences can be on the background rather than any object of interest.
- Moreover, many existing techniques are designed for *specific* analytic tasks, streaming media and/or filtering knobs. For instance, DDS [42] and STAC [84] select which regions in a frame should be in high quality, but its logic cannot be directly extended to adapt non-video input data, such as LiDAR point cloud, depth map, or audio waves.

Our approach: Fast gradient-based adaptation. We present OneAdapt, a configuration adaptation system that improves on all three fronts. OneAdapt uses gradient ascent to continuously tune the filtering knobs to improve the tradeoff between accuracy and resource usage. It uses a new technique to *cheaply* approximate the change of DNN inference accuracy in response to a small change on each filtering knob, which we refer to as ***AccGrad***.[1]

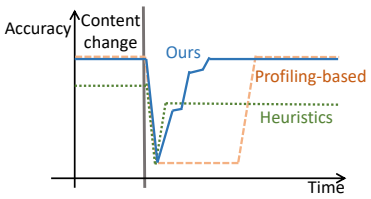


Figure 1: *Demonstrating the adaptive behavior of OneAdapt compared to alternatives: OneAdapt quickly adapts to a near-optimal configuration after the change of input content, whereas the profiling-based method adapts slowly and the heuristic-based method adapts quickly but suboptimally.*

The fast approximation of AccGrad is based on the observation that filtering knobs influence the inference accuracy through their changes on the DNN's input. Thus, the AccGrad of a filtering knob can be decoupled into two parts, both of which can be computed with low overheads.

1. *InputGrad:* How the DNN's input changes with each knob, which can be done on CPU without GPU compute.
2. *DNNGrad:* How the DNN's accuracy changes with its input, which can be obtained by a single DNN backpropagation.

We further speed up the existing backpropagation operator by not computing the gradients on DNN weights (§4 for more cost reduction techniques). Moreover, because DNNGrad describes DNN's sensitivity to its input change, regardless of which knob causes the input change, DNNGrad can be computed ***once*** and then be multiplied with the InputGrad of different knobs to get their AccGrad.

Since AccGrad can be approximated with minimum extra GPU compute, OneAdapt updates the estimate of AccGrad frequently (*e.g.,* every second) and runs a ***gradient-ascent*** strategy to update the configuration frequently in a way that maximally increases accuracy or reduces resource usage without hurting accuracy. Like in other similar settings [41], this ***gradient-ascent*** strategy is likely to converge to a near-optimal configuration, because the configuration-accuracy relationships are mostly *concave*.[2] §3 offers formal and intuitive explanation why AccGrad can be decoupled (§3.3) and why OneAdapt converges (§3.5).

[1]Note that we define AccGrad as *numerical* gradient rather than analytical gradient, which is used [35, 41, 73] when optimizing discrete system knobs.
[2]The concavity can be intuitively explained as: if we increase the knob by a small amount (*e.g.,* increase frame rate), the gain in accuracy is more significant when the system is of low accuracy, but the gain diminishes when the system is already generating accurate inference results.

To put OneAdapt's contribution into perspective, using DNN gradient is not new in video analytics systems [84, 93], but to the best of our knowledge, OneAdapt is the first to enable fast approximation of accuracy's gradient with respect to a range of popular filtering knobs. As illustrated in Figure 1, OneAdapt outperforms both profiling-based and heuristic-based approaches. Unlike profiling-based methods, we estimate AccGrad and adapt more frequently (by default, every second), while profiling-based methods adapt configuration after the slow profiling finishes (*e.g.,* every minute [89]). Unlike heuristics-based methods which either analyze the input data or the outputs of the DNN (including intermediate outputs [42, 67, 95]), OneAdapt can converge to a *closer-to-optimal* configuration as AccGrad directly indicates how DNN accuracy varies with a change in the configuration.

We evaluate OneAdapt on nine streaming-media analytics pipelines (Table 3) covering four analytic tasks, five types of input data and five filtering knobs. We compare OneAdapt with the latest profiling-based or heuristic-based schemes designed for individual knob types. Our key results are:

1. OneAdapt reaches similar accuracy while reducing bandwidth usage or GPU compute by 15-59%. Alternatively, OneAdapt improves accuracy by 1-5% without using more bandwidth usage or GPU usage compared to the baselines.
2. Compared to a straightforward implementation, OneAdapt reduces the GPU computation overhead and GPU memory overhead of AccGrad estimation by 87% and 12%, keeping OneAdapt's extra computation overhead below 20% of DNN inference.
3. Unlike solutions that are designed for specific filtering knobs, OneAdapt achieves its improvement across all filtering knobs using the same adaptation logic.

That said, OneAdapt still has its limitations (§7), such as not handling non-filtering knobs (*e.g.,* DNN selection) and tasks outside streaming media analytics (*e.g.,* generative tasks).

This work does not raise any ethical issues.

2 BACKGROUND

2.1 DNN-based streaming media analytics

DNN-based analytics applications are ubiquitous [1, 6, 20, 23, 24, 50, 59, 69, 78–80]. In this paper, we focus on streaming media analytics, where the analytic tasks (*e.g.,* detection and segmentation) are performed on streaming media (*e.g.,* RGB video, LiDAR video, audio, etc). Streaming media analytics are widely used in applications like autonomous driving (which requires object detection on RGB and LiDAR video), human detection (which requires human detection on InfraRed video at night), and smart home (which requires word detection on audio data). Note that streaming media analytics include popular video analytics applications though not all DNN-based applications (*e.g.,* generative AI).

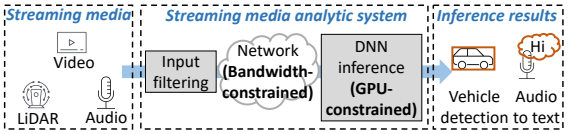


Figure 2: *In streaming media analytics, data from sensors is processed by the **input filtering** module before being sent to the **DNN** on constrained GPU resources. Given the DNN module might be on a remote device or cloud, data transmission can occur over a bandwidth-limited network.*

Typically, a streaming media analytic system, depicted in Figure 2, consists of *input filtering* (typically at the sensor side) and *DNN inference* (typically in edge or cloud). First, the system collects the raw data, referred to as the *input data*. This data is then filtered by the input filtering process. When the data needs to be sent to a separate analytics server (in edge or cloud), it will be streamed through a *bandwidth-constraint* link. The server then runs DNN inference to get inference results (*e.g.,* vehicle bounding boxes for autonomous driving) using *limited* GPU resources. Due to limited GPU resources and network bandwidth, not all data in their highest fidelity will be analyzed by the DNN.

Objective: The objective of these systems is to reduce resource usage (in GPU and bandwidth) without hurting the inference accuracy, or improve inference accuracy using limited resources. Following prior work [28, 42, 43, 61, 86, 89], we define *inference accuracy* of an inference result as its similarity with the inference result obtained under unlimited bandwidth and/or GPU computation budget (see §5.1 for detailed definition). This definition of accuracy highlights the impact of saving resources on inference accuracy.

2.2 Configuration adaptation

Prior work has shown that high accuracy *could* be obtained with significantly lower network and/or GPU usage if an *optimal configuration* of the filtering parameters (*e.g.,* video resolution or frame rate) is chosen. We denote these filtering parameters as *knobs*. Moreover, the optimal configuration of these knobs can be *highly sensitive* to the content of the input data [43, 61, 89, 92]. For instance, when detecting vehicles in a traffic video, we can significantly reduce the frame rate if the vehicles are moving slowly, and the DNN can still accurately determine the position and movement of the vehicles.

Depending on how knobs filter the input data, Table 1 categorizes the knobs along two dimensions.

- *Spatial vs. temporal:* A knob can filter (downsample) input data spatially (*e.g.,* lowering video resolution) or temporally (*e.g.,* reducing frame rate).
- *Coarse-grained vs. fine-grained:* A knob, spatial or temporal, can filter the input data with a coarse or fine granularity.

Importantly, depending on the type of knob, its optimal configuration can be sensitive to different aspects of the input

Knobs	Spatial vs. Temporal	Coarse-grained vs. Fine-grained
Resolution [34, 42, 61, 74, 88, 89, 91, 92]	Spatial	Coarse-grained
Quantization parameter (QP) [88, 89, 92]	Spatial	Coarse-grained
Frame rate [61, 74, 88, 89, 91]	Temporal	Coarse-grained
Frame filtering thresholds [32, 66, 94]	Temporal	Fine-grained
Region-based QP [21, 42, 43, 64, 67, 93]	Spatial	Fine-grained
Audio sampling rate [46, 55, 56, 58, 70]	Temporal	Coarse-grained

Table 1: *A list of filtering-related knobs that could be used to reduce bandwidth usage and/or GPU compute without hurting inference accuracy.*

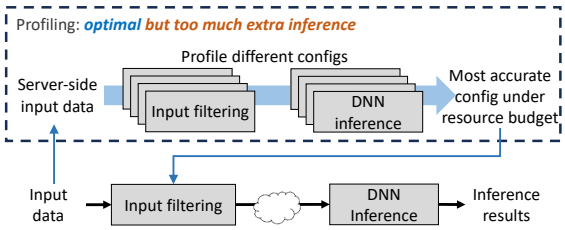
Adaptation methods	Knob types			
	Coarse-grained spatial	Fine-grained spatial	Coarse-grained temporal	Fine-grained temporal
Profiling [61, 88, 89, 91]	✔		✔	
Pixel-filtering heuristics [42, 43, 67, 84, 94]		✔		
Frame-filtering heuristics [32, 66]				✔
Uniform-filtering heuristics [92]	✔		✔	
OneAdapt (this work)	✔	✔	✔	✔

Table 2: *Unlike prior work that works well only for specific knobs, OneAdapt can optimize all four types of knobs in Table 1.*

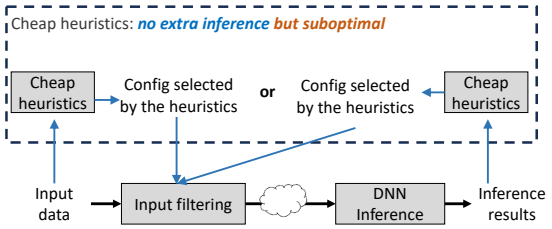
data. For instance, when filtering a video for object detection, a *spatial* and *coarse-grained* knob can lower the resolution of a whole video frame, and its optimal configuration depends on the *size* of the objects. A *spatial* but *fine-grained* knob can lower the resolution only in certain regions in a video frame, and its optimal configuration depends on the *location* of the objects. Similarly, a *temporal* and *coarse-grained* knob can uniformly reduce the video frame rate to fit *how fast* objects move, whereas a *temporal* but *fine-grained* knob can decide whether each frame should be dropped individually to fit *when* objects move.

2.3 Existing adaptation schemes

As the content of input data varies over time, these configurations must be adapted timely. For instance, it has been shown that increasing the configuration adaptation frequency of frame rate and video resolution from once every 8 seconds to once every 4 seconds increases the percentage of frames with accurate object detection results by 10% [61]. Other works also frequently adapt the configurations of other knobs, such as the encoding quality of each spatial region, at a time scale of a few seconds [42, 43, 67, 92].



(a) *Profiling: optimal but too much extra inference.*



(b) *Heuristics: no extra inference but suboptimal.*

Figure 3: *Illustrating two types of prior work: profiling and heuristics. The profiling-based approach can obtain optimal configuration but runs a lot of extra DNN inferences, while heuristics run no extra DNN inference but may pick suboptimal configuration (§2.3).*

Therefore, the key research question is to identify if a different configuration is better than the current one. There are two high-level categories of techniques.

Profiling-based methods: optimal though slow. The first approach *periodically profiles* resource usage and inference accuracy of different configurations by rerunning the same input data using these configurations, and then picking the best configuration (one that yields high inference accuracy and low resource usage) [61, 88, 89, 91]. Figure 3a illustrates this process. However, each profiling needs to run multiple *extra* DNN inferences and thus significantly increases the GPU computation overhead. For instance, AWStream [89] runs *full* profiling on just 3 knobs (resolution, frame rate, and quantization parameter) on a *10-second* video, and its GPU computation is equivalent to running normal DNN inference on more than an *8-minute* video (*i.e.,* an almost 50× increase in GPU usage). As a result, if GPU resource is limited, this approach must either profile less frequently, causing it to use outdated configurations when the optimal configuration changes (as shown in Figure 1), or profile only fewer configurations, which may miss the optimal configuration. §5 will also test smarter variants of profiling (profiling only top k configurations and profiling fewer values per knob) and shows that they still use much GPU resource for extra inference and are thus slow to adapt to change in input data.

No extra inference though suboptimal: An alternative approach [29, 32, 42–44, 66, 67, 84, 92, 94, 95] avoids extra

inference and instead selects new configurations by analyzing either the input data or the DNN intermediate inference results and filtering out those data that are not of the interest of the final DNN using simple heuristics. We illustrate this process in Figure 3b. This approach can adapt frequently as it does not require extra inference by the final DNN. However, they may select a suboptimal configuration. Here, we discuss three types of heuristics.

- *Heuristics adapting fine-grained temporal knobs:* One line of work [32, 66, 67] selects the frames most worthy for analysis by applying fine-grained temporal knobs (*e.g.,* by computing the difference between the current frame and previously analyzed frame and sends the current frame out for DNN inference if the difference is greater than a threshold). As elaborated in §1, this method fails when the background pixels in a video change a lot between frames, making high frame difference a poor signal to decide if a frame should be analyzed or not.
- *Heuristics adapting fine-grained spatial knobs:* Another line of work [42–44, 67, 94] reduces bandwidth usage by calculating the regions of interest, and assigns these regions with higher encoding quality. Some of these work identifies regions of interest by running a *shallow neural network* on each frame (*e.g.,* MobileNet-SSD [43, 57]). However, as acknowledged in one of the latest work [43], such shallow neural network rarely recognizes small vehicles and thus fails to encode them in high quality. Other heuristics try to leverage information naturally emitted from the final DNN, such as region proposals (an intermediate output of some DNNs indicating which regions in the input data may contain objects). This approach can also adapt quickly, but as shown in [43], region proposals only indicate whether *some* objects are in the region, rather than whether raising the quality of that region will likely affect the ability for the DNN to detect the objects of interest.
- *Heuristics adapting coarse-grained knobs*: Recent proposals also propose heuristics based on reinforcement learning [92] to adapt coarse-grained knobs such as resolution and frame rate. This heuristic relies on the file size of the encoded video to adapt the knobs. However, the file size of the encoded video cannot indicate whether the object of interest appears in the video or not, making this approach unable to timely raise the frame rate and resolution when the object of interest appears.

Bayesian Optimization: Though less explored in the literature of streaming media analytics systems, an alternative method can treat the analytic system as a black box and use Bayesian Optimization (BO) to search for the best configuration [15, 83]. In contrast, gradient-based optimization, which our work belongs to, has been shown to converge faster than BO in a discrete configuration space with a concave objective function. In theory, after evaluating k configurations, gradient-based optimization's function value has a gap-to-optimal of $O(\frac{1}{k^2})$ [71, 77] whereas BO's gap-to-optimal is $O(\frac{1}{\sqrt{k}})$ [39, 65]. As a result, gradient-based optimization, rather than BO, has been widely used in various discrete spaces with concave functions (*e.g.,* [35, 36, 73, 90]).

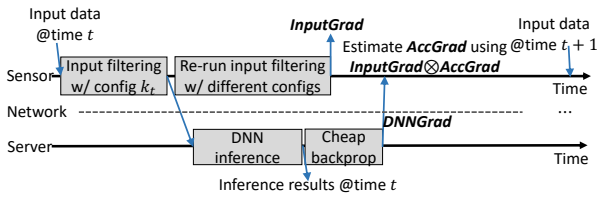


Figure 4: *Illustrating how OneAdapt estimates AccGrad using InputGrad and DNNGrad using the sensor and the server.*

In the case of streaming media analytics, the configuration values are discrete and, as shown later in §3.5, the objective function between the configuration and accuracy is mostly concave. Thus, we choose to use gradient-based optimization, rather than BO, and utilize DNN's differentiability to estimate the (numerical) gradients without extra DNN inference.

Designed for specific configurations or DNNs: Moreover, most techniques are studied and evaluated *only* on RGB video feeds, and it is unclear whether they will be effective on other types of data (*e.g.,* LiDAR point cloud, depth map, or audio waves). Even in video analytics, the heuristics often work well only for *specific knobs* (we show the applicability of prior works on different types of knobs in Table 2). For example, Reducto [66]'s frame-different detector is ill-suited to tune the encoding quality parameter of each frame, which depends on the content of the new frame, rather than how it differs from the previous frame. Likewise, DDS [42] tunes the encoding quality parameters of each fine-grained spatial block to improve inference accuracy on each frame, but it is not aware of whether the current frame is redundant given past inference results, thus it cannot generalize to knobs such as frame selection. Also, as the amount of extra inference scales with the number of knobs, profiling-based approach [61, 88, 89, 91] is suboptimal when handling more than 10 knobs (which is the case for fine-grained knobs).

Summary: In short, previous work fails to meet at least one of the three requirements: *(i)* adapt timely, *(ii)* converges to a near-optimal configuration, and *(iii)* apply to a wide range of configurations and applications.

3 DESIGN OF ONEADAPT

We present OneAdapt, a configuration adaptation system that aims at improving along the three requirements mentioned above — frequent, near-optimal adaptation on various configuration knobs. The basic idea of OneAdapt is to frequently estimate the gradient of how inference accuracy changes with each configuration knob, which we refer to

as AccGrad. OneAdapt then performs gradient ascent based on AccGrad to update the configuration, with an objective metric that combines inference accuracy and resource usage (§3.2). To efficiently estimate AccGrad, OneAdapt approximates it by estimating OutputGrad, another metric that can be efficiently calculated by leveraging DNN's inherent differentiability (§3.3) and statistically correlated with AccGrad (§3.4). Finally, we will discuss the caveats of OneAdapt's gradient-based adaptation (§3.5).

3.1 Terminology

To adapt configuration for various DNN pipelines , we introduce a unified terminology to describe their input and output. Table 4 summarizes the notations and their meanings.

We split the input data into fixed-length (by default, one-second-long) intervals, called *adaptation intervals*. In the t^{th} adaptation interval, the input data $\mathbf{x}_t$ is first preprocessed with the current configuration $\mathbf{k}_t$ (*e.g.,* video resolution and frame rate) to get the DNN input $\mathbf{y}(\mathbf{k}_t;\mathbf{x}_t)$, and the DNN takes it as input and returns $Res(\mathbf{k}_t;\mathbf{x}_t)$ as output. DNN output $Res(\mathbf{k}_t,\mathbf{x}_t)$ contains multiple *elements*. For the DNNs in our considered tasks, each element is associated with a confidence *score*, and only elements with a score over a confidence *threshold* θ will be counted towards accuracy.

The definitions of an element e and input data $\mathbf{x}$ vary with the considered application. An element is a detected object in object detection, a detected text in audio-to-text translation, or a segmented mask of a detected instance in instance segmentation. Input data can be a sequence of RGB frames in videos, point-cloud frames in LiDAR, or an audio wave segment in audio data.

Based on these notations, we define:

- $Acc(\mathbf{k}_t;\mathbf{x}_t)$ is the accuracy of the inference results generated using input data $\mathbf{x}_t$ and configuration $\mathbf{k}_t$. Following prior work [26, 28, 42, 43, 61, 89], we define accuracy as the ***similarity*** between the current inference results and the inference results generated from the most resource-consuming configuration $\mathbf{k}^*$.[3]
- $\mathbf{z}(\mathbf{k}_t;\mathbf{x}_t)$ is the output utility of the inference results generated using input data $\mathbf{x}_t$ and configuration $\mathbf{k}_t$. We define output utility as the number of elements with confidence scores above the confidence threshold (see §3.3).
- $\mathbf{r}(\mathbf{k}_t)$ is the resource usage (bandwidth and/or GPU cycles, defined per application) at the t^{th} adaptation interval.

We may omit the time label t and the input data $\mathbf{x}_t$ for simplicity when the corresponding variables are of the same interval as the current input data $\mathbf{x}_t$.

[3]This notion of accuracy is well studied in the literature. While it does not compare DNN output with human-annotated ground truth, it better captures the impact of adapting configurations on inference results.

3.2 Adaptation goal and gradient-ascent

The adaptation goal of OneAdapt is to pick the configurations for all adaptation intervals such that the following weighted sum between the accuracy Acc and the resource usage $\mathbf{r}$ is maximized:

$$\sum_{t=1}^{T} \underbrace{Acc(\mathbf{k}_t)}_{\text{accuracy of } t^{th} \text{ interval}} - \lambda \underbrace{\mathbf{r}(\mathbf{k}_t)}_{\text{resource usage of } t^{th} \text{ interval}}, \quad (1)$$

where λ is a hyperparameter that governs the tradeoff between accuracy and resource usage: higher λ will emphasize resource-saving and lower λ will emphasize accuracy improvement. We will vary λ and examine its effect on OneAdapt in §5. Note that this objective can be extended to perform *budgeted* optimization (*e.g.,* to maximize accuracy under a budget of resource usage, we can add a change the term $\mathbf{r}(\mathbf{k}_t)$ to a large penalty when resource usage extends the budget). We leave this extension to future work.

To optimize towards this goal, OneAdapt uses an online ***gradient-ascent strategy*** (illustrated in Figure 4). Concretely, OneAdapt derives a new configuration $\mathbf{k}_{t+1}$ based on the current configuration $\mathbf{k}_t$ using the following Equation:

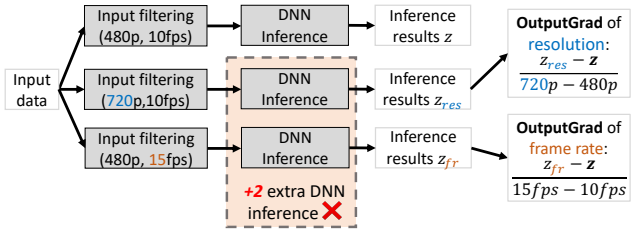
$$k_{t+1,i} = k_{t,i} + \alpha \left(\underbrace{\frac{Acc(\mathbf{k}_t + \Delta k_i) - Acc(\mathbf{k}_t)}{\Delta k_i}}_{\text{grad. of accuracy (\textbf{AccGrad})}} - \lambda \underbrace{\frac{\mathbf{r}(\mathbf{k}_t + \Delta k_i) - \mathbf{r}(\mathbf{k}_t)}{\Delta k_i}}_{\text{grad. of resource usage}} \right), \quad (2)$$

where $k_{t+1,i}$ is the configuration of the i^{th} knob in the next adaptation interval, Δk_i is a small increase on the i^{th} knob, α is the learning rate. We will discuss the convergence of this gradient-ascent strategy and extend it to discrete configuration values in §3.5. For now, we assume the configuration of each knob can be tuned continuously. Also, while more advanced gradient-based optimization (*e.g.,* Nesterov's Accelerated Gradient Descent [71]) exists, OneAdapt chooses a standard gradient-ascent strategy to make sure the source of improvement is from the gradient itself instead of advanced optimization techniques.

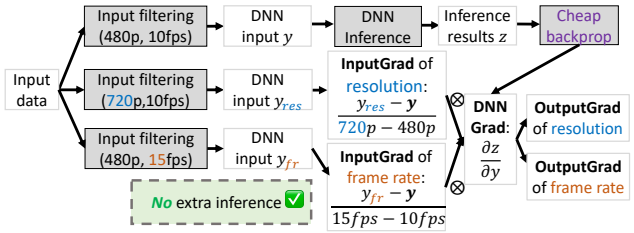
The gradient-ascent strategy in Equation 2 requires calculating the gradient of accuracy (hereinafter AccGrad) and the gradient of resource usage along each knob i. We calculate the gradient of resource usage by using its definition (*i.e.,* we directly calculate the bandwidth and GPU computation consumption of configuration $\mathbf{k}_t + \Delta k_i$ and then calculate $\frac{\mathbf{r}(\mathbf{k}_t+\Delta k_i)-\mathbf{r}(\mathbf{k}_t)}{\Delta k_i}$)[4].

However, if we calculate AccGrad by its definition, it would be prohibitively expensive (see Figure 5a). Obtaining $Acc(\mathbf{k})$ of any configuration $\mathbf{k}$ will require running inference

[4]Note the calculating the resource usage of a configuration does not require DNN inference, as calculating the bandwidth usage requires no inference, and the GPU computation largely depends on the shape of DNN input rather than the content of the input.



(a) Calculating OutputGrad using its definition



(b) Calculating OutputGrad using OneAdapt

Figure 5: *Comparison between calculating OutputGrad naively and calculating OutputGrad using OneAdapt. OneAdapt calculates OutputGrad with* ***no extra inference****.*

on the most resource-intensive configuration $\mathbf{k}^*$. Moreover, to get the AccGrad of each knob would require running an extra inference to test the impact of a small change in configuration on DNN output.

3.3 Fast approximation of AccGrad

To estimate AccGrad efficiently, we introduce a new metric, called OutputGrad, to approximate it. Here, we define OutputGrad and explain how to calculate it efficiently. The next subsection will show its statistical correlation with AccGrad.

DNN output utility: Recall from §3.1 that the DNN output contains multiple *elements*, each associated with a confidence *score*. An element can be a detected object in object detection or a detected text in audio-text translation. Since only elements with a score over the confidence *threshold* will be counted towards accuracy, we define *output utility* of a DNN output by the number of elements in an adaptation interval whose confidence scores exceed the confidence threshold (see Equation 4 for a formal definition).[5] We use $\mathbf{z}(\mathbf{k}_t)$ to denote the output utility of DNN output $Res(\mathbf{k}_t; \mathbf{x}_t)$ under configuration $\mathbf{k}_t$.

Output utility offers a single-value summarization of the DNN output whose change indicates the change in inference accuracy (More rationale of output utility in §3.4).

OutputGrad definition: OutputGrad is defined as how much a small change in the current value of each knob changes the output utility. The OutputGrad of knob k_i can be written as $\frac{\Delta \mathbf{z}}{\Delta k_i} = \frac{\mathbf{z}(...,k_{t,i}+\Delta k_i,...) - \mathbf{z}(...,k_{t,i},...)}{(k_{t,i}+\Delta k_i) - k_{t,i}}$.

To help understand OutputGrad, we use a simple video analytics system that feeds each video frame to a vehicle detection DNN, using a confidence score of 0.5. Figure 5a illustrates OutputGrad in this example. Following prior work [61, 88, 91, 92], we assume the system can tune two configuration knobs, frame rate, and resolution. Suppose that the current configuration is 10 frames per second (fps) and 480p. When taking 1-second input, the DNN outputs 34 vehicle bounding boxes with confidence scores greater than 0.5 on these 10 frames, an average of 3.4 vehicles per frame. If we slightly increase the frame rate from 10fps to 15fps, the DNN outputs 66 vehicle bounding boxes with confidence scores greater than 0.5, on average 4.4 vehicles per frame. Then the OutputGrad of the current frame rate is $\frac{\Delta \mathbf{z}}{\Delta \text{framerate}} = \frac{\mathbf{z}(480p,15fps) - \mathbf{z}(480p,10fps)}{15fps - 10fps} = \frac{4.4-3.4}{15-10} = 0.2$. Similarly, the OutputGrad of the resolution is: $\frac{\Delta \mathbf{z}}{\Delta \text{resolution}} = \frac{\mathbf{z}(720p,10fps) - \mathbf{z}(480p,10fps)}{720p - 480p} = \frac{5.8-3.4}{720-480} = 0.01$.

How to calculate OutputGrad efficiently: The insight of OneAdapt is that OutputGrad can be computed efficiently *with no extra DNN inference* by taking advantage of the ***differentiability of DNNs***. As our considered application pipelines affect the inference results by altering the DNN input through knobs, the OutputGrad of a knob can be written as the inner product of two separate gradients (illustrated in Figure 4), each can be computed efficiently:

1. *DNNGrad:* How the DNN's output changes with respect to the DNN's input.
2. *InputGrad:* How the DNN's input changes with respect to the knob's configuration.

Formally, the OutputGrad of k_i on $\mathbf{x}$ can be expressed by: $\underbrace{\frac{\Delta \mathbf{z}}{\Delta k_i}}_{\text{OutputGrad}} = \underbrace{\frac{\Delta \mathbf{z}}{\Delta \mathbf{y}(\mathbf{k})}}_{\text{DNNGrad}} \otimes \underbrace{\frac{\Delta \mathbf{y}(\mathbf{k})}{\Delta k_i}}_{\text{InputGrad}}$, where $\otimes$ means inner product, and $\mathbf{y}(\mathbf{k})$ is the DNN input of configuration $\mathbf{k}$.

The decoupling makes the calculation of OutputGrad much more resource-efficient for three reasons:

- DNNGrad can be estimated by running one backpropagation . This is a constant GPU-time operation without extra inference as it reuses the layer-wise features computed as part of the DNN inference of the current configuration. §4 will further reduce the overhead of backpropagation.
- DNNGrad only needs to be calculated once and can be reused to derive OutputGrad of different knobs without extra DNN inference. This is because the DNNGrad describes DNN's sensitivity to its input and thus remains largely similar if the change in DNN input is not dramatic. For instance, the DNNGrad of an object-detection DNN is high on pixels associated with key visual features of an object.

[5] This definition applies to all applications mentioned in Table 3, but future work is needed to extend the definition to a wider range of applications.

- Finally, computing InputGrad does not require running the final DNN.

To reuse the same example from Figure 5a, Figure 5b shows how OutputGrad is calculated from DNNGrad and InputGrad:

$$\frac{\mathbf{z}_{(480p,15fps)} - \mathbf{z}_{(480p,10fps)}}{15fps - 10fps} \approx \frac{\mathbf{y}_{(480p,15fps)} - \mathbf{y}_{(480p,10fps)}}{15fps - 10fps} \otimes \frac{\Delta \mathbf{z}}{\Delta \mathbf{y}}\Big|_{\mathbf{y}=\mathbf{y}_{(480p,10fps)}},$$

$$\frac{\mathbf{z}_{(720p,10fps)} - \mathbf{z}_{(480p,10fps)}}{720p - 480p} \approx \frac{\mathbf{y}_{(720p,10fps)} - \mathbf{y}_{(480p,10fps)}}{720p - 480p} \otimes \frac{\Delta \mathbf{z}}{\Delta \mathbf{y}}\Big|_{\mathbf{y}=\mathbf{y}_{(480p,10fps)}},$$

where $\otimes$ represents inner product and $\frac{\Delta \mathbf{z}}{\Delta \mathbf{y}}|_{\mathbf{y}=\mathbf{y}_{(480p,10fps)}}$ is DNNGrad of the current configuration $(480p, 10fps)$. Figure 5b shows that OneAdapt saves extra GPU inference by running backpropagation once and re-using the backpropagation results for different knobs. This example has two knobs, thus the saving may seem marginal, but our evaluation shows more savings when OneAdapt is used to adapt more knobs in more realistic DNN analytics systems (Figure 9).

3.4 Relationship between OutputGrad and AccGrad

The definitions of OutputGrad and AccGrad differ, yet they are closely related both theoretically and empirically This can be intuitively explained as follows: a high OutputGrad means a great change in the inference output, which often causes a greater change in accuracy and thus a high AccGrad.

Theoretical correlation: To formalize this correlation between OutputGrad and AccGrad, we prove that they are statistically correlated under specific ***assumptions***.

- First, we define the accuracy of DNN output as the number of correctly-identified elements (including both true positive and true negative), minus the number of wrongfully-identified elements (in our proof we use a differentiable approximation of this accuracy function, see Equation 3 for formal definition). Acknowledging that this definition of accuracy differs from the metric that is widely used in prior work (*e.g.,* F1 score [28, 42, 43, 61, 67, 88, 89, 91, 92]), we observe that, if a configuration has higher accuracy than another configuration under one accuracy metric, it is likely that this configuration also has higher accuracy under other accuracy metrics.
- Second, the accuracy of DNN output increases when a knob changes towards using more resources (*e.g.,* higher resolution or selecting more frames). This observation aligns with prior work [26, 42, 43, 61, 66, 67, 88, 89, 91].
- Third, different elements do not overlap (*e.g.,* object bounding boxes do not mutually overlap, or the detected words in the audio do not overlap). This assumption holds as a wide range of DNNs perform non-maximum suppression [40, 53, 72] to remove overlapped elements.

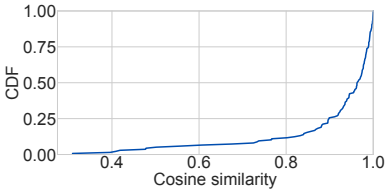


Figure 6: *The CDF of the cosine similarity between AccGrad and OutputGrad across different configurations and videos. The average cosine similarity is over 0.91.*

Formally, if these assumptions hold, we can formally prove that $\underbrace{\frac{\partial Acc(\mathbf{k})}{\partial \mathbf{k}}}_{\text{AccGrad}} = \Big|\underbrace{\frac{\partial \mathbf{z}(\mathbf{k})}{\partial \mathbf{k}}}_{\text{OutputGrad}}\Big|$. The proof can be found in §E.

While there can be exceptions to the conditions required by our proof, these conditions corroborate the observations in previous studies. For instance, higher encoding resolution leads to higher bandwidth usage and likely more accurate inference, or higher frame rate leads to higher GPU usage and likely more accurate results [26, 42, 43, 61, 66, 67, 88, 89, 91].

Empirical correlation: We empirically validate the correlation between AccGrad and OutputGrad using a streaming media analytics pipeline (pipeline ⓐ) from the Downtown dataset (refer to §5 for details). For each configuration in pipeline ⓐ , we derive AccGrad from its definition and OutputGrad through backpropagation on the first 10 seconds of each video in Downtown dataset. Since both AccGrad and OutputGrad are vectors (with each element being the gradient for a knob), we measure their correlation by cosine similarity. Figure 6 displays the CDF of cosine similarity across different configurations and videos. The average cosine similarity exceeds 0.91, indicating a strong correlation between AccGrad and OutputGrad.

3.5 Caveats and benefits

While our evaluation shows OneAdapt's gradient-ascent strategy performs well for a wide range of DNN tasks, input data types, and configuration knobs, it is important to understand its limits and why it works in practice.

Convergence of OneAdapt on changing input data: OneAdapt takes a gradient-ascent strategy to adapt configurations, where OneAdapt derives a new configuration based on the input data of the current adaptation interval and applies it to the next interval. This approach will not converge if the input changes dramatically between intervals. That said, the input data in our benchmark (and other papers [61, 89]) incurs drastic change on the scale of tens of seconds (*e.g.,* when driving in the countryside, the vehicle appears on a scale of tens of seconds), whereas OneAdapt empirically converges to a near-optimal configuration within 3-5 intervals (seconds) as shown in Figure 14. Thus, as long as the best configuration lasts for at least a few seconds,

OneAdapt can identify the best configuration faster than profiling-based methods.

The reason behind the convergence of OneAdapt is that the relationship between configuration $\mathbf{k}$ and its corresponding accuracy $Acc(\mathbf{k})$ is roughly ***concave***, allowing the gradient-ascent strategy of OneAdapt to converge to a near-optimal configuration. This concavity also appears in prior work [42, 43, 67, 89]. This concavity means that as configuration changes in the direction of using more resources, the improvement in accuracy will diminish as accuracy approaches 100%. For example, as a video analytic system increases the video encoding resolution, the pixel change becomes more marginal (mostly on "high-frequency" details), causing the DNN output to stabilize (objects' confidence scores will change more slowly and less frequently cross the confidence threshold, leading to fewer newly detected objects) and leading to less improvement on accuracy [61, 88, 89, 91, 92]. A well-known consequence of such concavity is that gradient-ascent can eventually converge to a global optimal configuration [10, 45].

Why OneAdapt beats alternatives: OneAdapt outperforms previous work on all three requirements (*i.e.,* adapt frequently, converge to a near-optimal configuration, and generalize to different types of knobs and analytic tasks).

First, compared to profiling-based work, OneAdapt adapts much more frequently (as OneAdapt updates its configuration *at every second* but profiling-based work updates its configuration once every minute [89]) and earlier (as OneAdapt adapts to the change of input content right at the next second but the profiling-based work need to wait till the next profiling).

Second, compared to heuristics-based methods, OneAdapt's gradient ascent can converge to a *closer-to-optimal* configuration. Most heuristics used in prior work adapt configuration by analyzing only the input data, rather than how the data might affect the final DNN's output and accuracy. By contrast, AccGrad directly indicates that DNN accuracy varies with a small change in the configuration.

Third, we show that OneAdapt can generalize to 4 analytic tasks (vehicle detection, human detection segmentation, audio-to-text), 5 types of input data (*e.g.,* LiDAR videos, audio waves), and 5 types of knobs (*e.g.,* video codec knobs, frame filtering thresholds) in our evaluation (§5). That said, we have not tested if OneAdapt will work for applications outside streaming media analytics (*e.g.,* text generation and image generation) and knobs that alter the final DNN itself (*e.g.,* DNN selection).

We want to clarify that similar to prior work [61, 89, 92], OneAdapt leverages idle time to perform adaptation. Thus, OneAdapt does not delay the generation of inference results.

Running gradient ascent over discrete configuration space: Before running OneAdapt, we first linearly normalize all knob values to [0,1] and make sure that the increase in knob value corresponds to the increase in resource usage. We then calculate the updated knob value (Equation 2) and configure each knob using the configuration value closest to the updated value. We want to clarify that running gradient ascent on top of discrete configuration space can converge to a near-optimal configuration when the objective function (*i.e.,* the weighted sum between accuracy and resource usage) is concave [35, 73].

4 OPTIMIZATION OF ONEADAPT

Though OutputGrad estimation in OneAdapt does not involve extra DNN inference, it imposes three types of overhead:

1. *GPU overhead* to run backpropagation and obtain DNNGrad.
2. *CPU overhead* to filter input data multiple times and compute InputGrad of each knob.
3. *Bandwidth overhead* to stream the DNNGrad from the DNN back to the sensor (as shown in Figure 4).

The first two overheads are present in both the distributed setting (where the sensor streams DNN input through a bandwidth-constraint link to a server for DNN inference) and non-distributed settings (where the sensor and DNN co-locate). The third overhead is specific to the distributed setting. This section presents the optimizations used by OneAdapt to reduce each overhead.

We want to clarify that similar to prior work [61, 89, 92], OneAdapt leverages idle time to adapt. Thus, these overheads of OneAdapt do not delay the generation of inference results.

4.1 Reducing GPU computation overhead

OneAdapt uses two techniques to reduce the GPU overhead of backpropagation.

Removing unneeded computation: The standard implementation of DNN backpropagation is designed for DNN training, which computes not only gradients with respect to the DNN input (*i.e.,* DNNGrad) but also gradients with respect to DNN weights. While the gradients with respect to DNN weights are crucial for DNN training (in order to update DNN weights), they are unnecessary to compute DNNGrad. We note that this optimization does *not* change DNNGrad.

DNNGrad reusing: To further reduce GPU computation overhead, we run backpropagation to compute DNNGrad on the last frame or segment of the current adaptation interval, and reuse this DNNGrad on other data frames in the current adaptation interval. In the context of video analytics, it can be intuitively understood that although the exact value of DNNGrad may vary across different frames, the *spatial* areas that have high DNNGrad tend to be stable within several consecutive frames. Though DNNGrad reusing alters DNNGrad,

Figure 12 shows that this optimization has little impact on the resource–accuracy trade-off of OneAdapt.

Summary: After removing unneeded computation and DNNGrad reusing, OneAdapt's GPU computation overhead is below 20% of DNN inference (as shown in Figure 11) and the GPU memory overhead is reduced by 12%.

4.2 Reducing CPU overhead

OneAdapt requires encoding the video multiple times to obtain InputGrad for each knob, incurring high CPU overhead.

OneAdapt reduces such overhead to one single encoding for those knobs that operate on *non-overlapping* spatial areas of DNN input (note that OneAdapt only uses this technique on a specific set of knobs like the encoding qualities of different spatial blocks and it may not generalize to other knobs.). For example, say the analytic system contains two knobs k_1, k_2, where k_1 is the encoding quality of the left half of the video, and k_2 is the encoding quality of the right half of the video. As a result, k_1 mainly changes the left half of the input video, and k_2 mainly changes the right half of the video[6]. To calculate the InputGrad of k_1 and k_2, We can change k_1 and k_2 *simultaneously* and then use the change of the left half of the input video to compute the InputGrad of k_1, and the right half to compute the InputGrad of k_2.

4.3 Reducing network overhead

OneAdapt imposes high network overhead as it streams the DNNGrad from the server back to the sensor. This overhead can be substantial. For instance, the DNNGrad of an object detector on a frame will be as large as the original raw, uncompressed frame size.

To compress DNNGrad, our observation is: modern codecs (*e.g.,* video codecs [38, 82]) typically partition the data into small groups, called *Minimum Coded Units* (MCUs), and then decide the compression scheme *on top of* these groups. In other words, the data inside each MCU will share the same compression scheme, thus there is no need to further distinguish between the data inside each MCU. Based on this observation, for each MCU, OneAdapt takes the absolute value of DNNGrad and averages the values of each data inside the MCU. For example, when the input video stream is encoded by H.264 [82] video codec, each MCU will be a 16×16 pixel block. OneAdapt then averages the DNNGrad values inside each 16×16 pixel block and obtains only one DNNGrad number for each pixel block, which compresses DNNGrad by $16^2\times$ and makes the bandwidth overhead of streaming DNNGrad negligible. We note that this optimization does not apply to all existing codecs, but it does cover a wide range of existing codecs (*e.g.,* video codecs [38, 82], image codecs [8] and audio codecs [12]) and it covers all codecs that OneAdapt is using in its evaluation.

[6] k_1 may slightly change the right half of the video. However, such change is minor compared to how much k_2 changes the right half of the video

5 EVALUATION

In our evaluation, we show that:

- OneAdapt achieves similar accuracy while reducing resource usage by 15-59% or improves accuracy by 1-5% with the same or less resource usage when compared to state-of-the-art adaptation approaches. This accuracy improvement is substantial, as the system needs to consume much more resources to improve accuracy by 1-5% (*e.g.,* 2x more bandwidth (Figure 7g) only improves 4% accuracy). This improvement is also on par with prior work [42, 43].
- The extra GPU overhead caused by OneAdapt is comparable to or lower than existing adaptation approaches.
- OneAdapt achieves more resource reduction when there are more configuration knobs to tune.

Our code is available in [9].

5.1 Evaluation setup

Applications and DNNs: We target four types of applications: vehicle detection with FasterRCNN for autonomous driving [40, 76], vehicle segmentation employing MaskR-CNN for traffic analytics [40, 54], human detection using YoLo for home security [19, 75] and audio-to-text utilizing Wav2Vec for smart home applications [7, 27]. Note that both detection and segmentation applications categorize objects as either of interest or not.

Accuracy metrics: We use F1 score [42, 61, 89, 92] for vehicle detection, vehicle segmentation, and audio-to-text. For human detection, we use mean IoU (mIoU [67]). Following prior work [26, 28, 42, 43, 61, 89], we define accuracy as the similarity between the current inference results and the inference results generated from the most resource-consuming configuration to measure the impact of adapting configurations on inference results.

Input settings: For all our applications we use 10FPS from the respective sensors (except for audio-to-text, where we chunk the raw audio into one-second segments and send them to the DNN for analytics). Note that while 30FPS and 60FPS are typical for video content intended for human viewing [3], 10FPS is prevalent in real-time analytic applications such as autonomous driving [5, 22, 48, 85].

Dataset: We use the following datasets to evaluate OneAdapt, with the goal of covering various application scenarios and streaming media (summarized in Table 5):

- *Autonomous driving*: our dataset covers two driving contexts (downtown driving and country driving) and two main types of sensors (RGB video sensor and LiDAR sensor). Specifically, we obtain 10 downtown driving RGB videos [17] and 8 country driving RGB videos [17] using an anonymous YouTube search, and 6 urban driving LiDAR videos from KITTI dataset [47].

ID	Target application	Analytic task	Dataset	Streaming media type	DNN	Accuracy metric	Configuration knobs	Trade-off between accuracy and	Baselines	Showing the generalizability of OneAdapt across
ⓐ	Autonomous driving	Vehicle detection	Downtown [17]	RGB	Faster-RCNN	F1 score	Fine-grained encoding quality assignment	Bandwidth usage	DDS [42] EAAR [67] AccMPEG [43]	Different **configuration knobs**
ⓑ							Video codec knobs		Chameleon [61] CASVA [92] AWStream [89]	
ⓒ			Country [17]				Frame filtering knobs	GPU computation	Profiling [61, 89] Reducto [66]	
ⓓ	Autonomous driving	Vehicle detection	KITTI [47]	LiDAR	Point-Pillars	F1 score	Fine-grained encoding quality assignment	Bandwidth usage	Region-based [42, 67] Uniform quality	Different types of **streaming media**
ⓔ	Home security	Human detection	PKU-MMD [37]	RGB, Depth, InfraRed	Yolo	mIoU	Fine-grained encoding quality assignment on DNN features	Bandwidth usage	Region-based [42, 67]	
ⓕ										
ⓖ										
ⓗ	Smart home	Audio-to-text	Google AudioSet [2]	Audio	Wav2Vec	F1 score	Audio sampling rate	Bandwidth usage	Profiling [61, 89] Voice detection [60]	Different **analytic tasks**
ⓘ	Traffic analytics	Vehicle segmentation	Traffic [17]	RGB	Mask-RCNN	F1 score	Fine-grained encoding quality assignment	Bandwidth usage	DDS [42] AccMPEG [43]	

Table 3: *Overview of the experimental setup. Each row represents an analytic pipeline we used to evaluate OneAdapt.*

- *Traffic analytics:* We collect 5 traffic camera video footag by anonymous YouTube searche [17].
- *Home security:* We use three types of sensor data (RGB video sensor, InfraRed sensor, and depth sensor), with 10 videos each from PKU-MMD dataset [37]. The inclusion of InfraRed and depth data is vital for enhancing night-time human detection accuracy in home security applications.
- *Smart home:* We randomly sample 200 audio clips in Google AudioSet [2].

Pipelines: We use nine analytic pipelines (pipeline ⓐ -ⓘ). These analytic pipelines show the applicability of OneAdapt across configuration knobs (pipeline ⓐ -ⓒ), types of streaming media (pipeline ⓓ -ⓖ) and applications (pipeline ⓗ -ⓘ). Table 3 summarizes these pipelines, including their target applications, analytic tasks, datasets, streaming media types, DNNs, and accuracy metrics.

Knobs and baselines: We describe the knobs and the corresponding baselines of these pipelines one by one:

- Pipeline ⓐ : This pipeline saves bandwidth by adjusting the encoding quality within each 16x16 pixel macroblocks [43, 82]. We benchmark against three methods. DDS [42] uses a low-quality video for initial inference, then refines specific regions with high-quality encoding. EAAR [67] uses results from the previous frame to identify current frame regions needing high-quality encoding. AccMPEG [43] runs a sensor-side neural network to determine regions for high-quality encoding.
- Pipeline ⓑ : This pipeline optimizes bandwidth using three knobs: resolution, QP, and B-frame selection likelihood [29, 32, 42–44, 61, 66, 67, 88, 89, 91, 92, 94]), all supported in prevalent video codecs [25, 38, 82]. We implement three baselines for this pipeline. Chameleon [61] periodically profiles top-k configurations and picks the one with the highest accuracy under the current bandwidth budget. AWStream [89] periodically profiles a downsampled subset of configurations and chooses the one that maximizes accuracy within bandwidth limits. CASVA [92] runs a reinforcement learning model on the sensor to determine the new configuration.
- Pipeline ⓒ : This pipeline reduces the GPU computation usage by running frame filtering using two frame filtering knobs (pixel difference threshold and area difference threshold [66]). As for baselines, we implement Reducto [66], together with a profiling-based baseline [89].
- Pipeline ⓓ : This pipeline saves bandwidth when streaming LiDAR point clouds by segmenting the 3D space around the LiDAR sensor into spatial blocks and streaming k% of LiDAR points from each. The value of k can vary between blocks. While this encoding mechanism is basic, our goal is not to establish a best practice, but to showcase the advantage of configuration adaptation. To ensure robustness, we average results from five repeated experiments. We compare against two baselines: a region-based method extended from [67] and a uniform-quality approach, which applies a fixed encoding quality across blocks.
- Pipeline ⓔ ⓕ ⓖ : These pipelines transform videos into feature vectors using a sensor-side DNN and then stream them for server analysis [21]. To reduce bandwidth usage, we vary encoding qualities across the spatial blocks of feature vectors. Traditional heuristics are not directly applicable on DNN-generated features, so we introduce a region-based heuristic that prioritizes blocks with recent human activity [42, 67]. Note that given the 16 configuration knobs in this pipeline, profiling methods fall short, even when configurations are aggressively downsampled. As evidence, we implement Chameleon [61] in pipeline ⓔ as the profiling baseline, which, despite 8x more GPU resources than OneAdapt and only tuning 4 knobs, still fails to outperform OneAdapt.

- Pipeline ⓗ : This pipeline optimizes the delay between a user's speech and its transcription into text by reducing bandwidth usage. We use the audio sampling rate as our knob. We implement a profiling-based baseline (Chameleon [61]) and a heuristics baseline that raises the audio sampling rate when detecting human voice [60].
- Pipeline ⓘ : This pipeline aims to reduce the bandwidth usage of running traffic analytics, by assigning different encoding quality to different spatial areas. We use the baselines same as pipeline ⓐ except for EAAR, as EAAR is not directly applicable to vehicle segmentation DNNs.

Resource usage: For those pipelines that aim to minimize bandwidth usage, we measure the bandwidth usage by the bandwidth needed to send one-second worth of data, while constraining the GPU computation as being able to analyze 1.5-second worth of data per second (we relax this constraint for one baseline (DDS [42]) as it needs to examine the same one-second data twice for each second). For those pipelines that aim to minimize GPU computation usage, we measure the GPU computation by the number of video frames that need to be analyzed per second[7]) and we do not constraint the bandwidth usage of OneAdapt and other baselines.

Hardware settings: We use one Intel Xeon 4100 Silver CPU as the CPU and NVIDIA RTX 2080 as the GPU.

5.2 Experimental results

Better trade-off between accuracy and resource: We show that OneAdapt achieves a better trade-off between accuracy and resource usage across 9 different pipelines in Figure 7. We see that across these applications, OneAdapt achieves 15-59% resource usage reduction compared to the baselines without decreasing inference accuracy, or improves the accuracy by 1-5% without inflating resource usage. We highlight that in Figure 7e, a profiling-based baseline (Chameleon) with 8x more GPU compute than OneAdapt still has a sub-optimal trade-off between resource usage and accuracy.

Adaptation behavior of OneAdapt: We compared the adaptability of OneAdapt against two baselines (heuristics-based and profiling-based) using the audio-to-text pipeline (pipeline ⓗ). As shown in Figure 8, up to the 42^{nd} second, there is no human voice and all methods use a low audio sampling rate while maintaining 100% accuracy. At the 42^{nd} second, the accuracy of all methods drops due to the continued use of the low sampling rate. OneAdapt then identifies a large AccGrad and promptly raises the sampling rate, achieving 100% accuracy at the 45^{th} second. In contrast, the profiling baseline persists with the outdated low sampling rate until its profiling completes at the 47^{th} second.

[7]The backpropagation overhead of OneAdapt is also transformed to the number of frames analyzed per second (by using backpropagation runtime divided by the runtime of analyzing one frame).

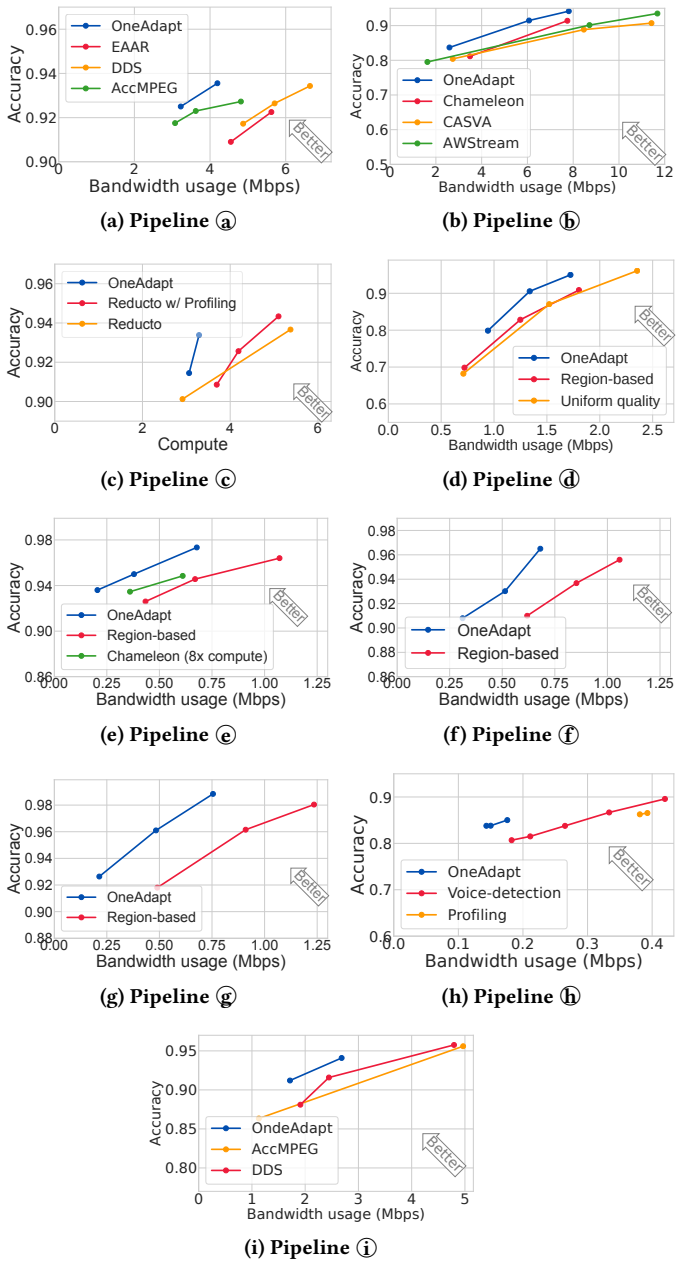


Figure 7: *Demonstrating the trade-off between accuracy and resource usage of of OneAdapt and several baselines. OneAdapt achieves higher accuracy with 15-59% resource usage reduction or 1-5% higher accuracy with less resource usage compared to the baselines on 9 different pipelines. We note that the results in each figure are averaged across 5-10 videos or 200 audios (depending on the dataset used for each figure) and thus have statistical confidence.*

The voice-detection heuristics, being overly cautious, uses a conservative sampling rate, resulting in suboptimal accuracy.

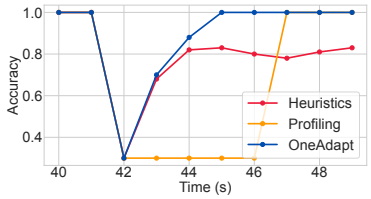


Figure 8: *Comparing the behavior of OneAdapt against the profiling-based baseline and heuristic-based baseline. The accuracy of OneAdapt and all baselines drop upon the person starts talking, but OneAdapt quickly adapts to the audio content change and improves its accuracy, while the accuracy of the baselines stays low for a while, as it profiles every 5 seconds.*

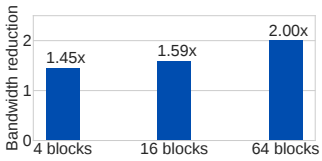


Figure 9: *When the knobs are more fine-grained, the bandwidth reduction of OneAdapt (the bandwidth usage of the region-based baseline, divided by the bandwidth usage of OneAdapt when OneAdapt is of higher accuracy) grows larger.*

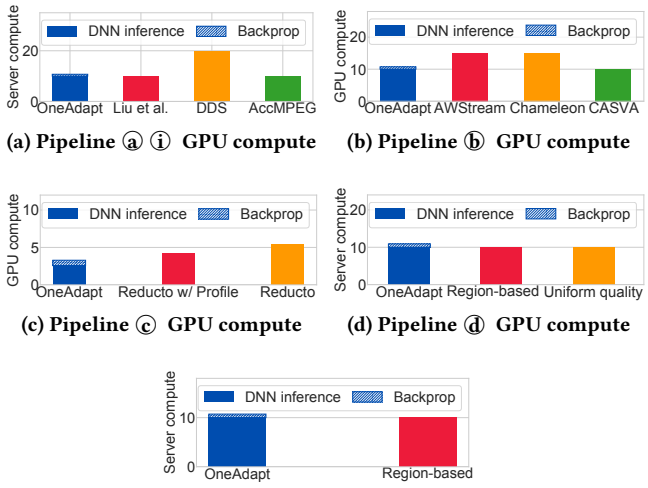


Figure 10: *Comparing the server-side compute overheads of OneAdapt against the baselines (measured by the number of frames inferred by the server per second).*

More knobs, more gain: We show that OneAdapt achieves higher bandwidth reduction[8] compared to the baseline in pipeline ⓔ . In Figure 9, encoding qualities are assigned to 4, 16, 64 spatial blocks, resulting in 4, 16, 64 knobs to adapt. We show that the bandwidth reduction of OneAdapt grows larger when there are more configuration knobs. This is because OneAdapt near-optimally handles more knobs without adding GPU computation (since OneAdapt only runs one backpropagation, regardless of the number of knobs) or CPU computation (by using the optimization for non-overlapping knobs in §4.2). However, the heuristics encode similar areas in high quality regardless of number of knobs and thus cannot significantly improve the resource–accuracy trade-off when there are more knobs.

[8]We define bandwidth reduction as the bandwidth usage of the region-based approach, divided by the bandwidth usage of OneAdapt when OneAdapt has higher accuracy

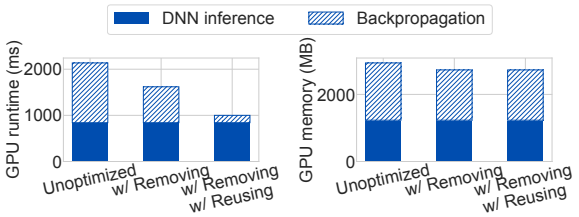


Figure 11: *Benchmarking the effectiveness of removing unneeded computation and DNNGrad reusing on a 10-frame video chunk using pipeline ⓐ . OneAdapt reduces the GPU runtime overhead by 87% and the GPU memory overhead by 12%.*

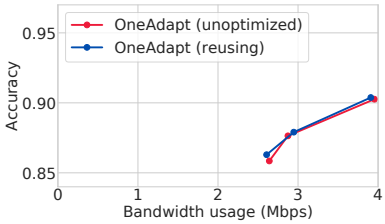


Figure 12: *Benchmarking the impact of DNNGrad reusing on pipeline ⓐ . DNNGrad reusing does not reduce the accuracy or increase the resource usage of OneAdapt.*

Overhead of OneAdapt: We measure the sensor-side CPU computation and the server-side GPU computation of OneAdapt[9] using the CPU computation divided by the CPU computation of processing one frame (or 1/10 worth of data in other data formats), and the GPU computation divided by the GPU computation of analyzing one frame. We mark the adaptation overhead of OneAdapt in the hatched area. From Figure 10, we see that the server-side GPU computation of OneAdapt is comparable to or lower than the baselines, and the adaptation overhead of OneAdapt is negligible.

That said, the sensor-side CPU overhead of OneAdapt is high. Though OneAdapt has equal or lower CPU overhead in pipeline ⓒ ⓗ than the baselines, OneAdapt needs to encode the input data 3 times (in pipeline ⓐ ⓓ ⓔ ⓕ ⓖ ⓘ , even 4 times in pipeline ⓑ), resulting in more CPU computation overhead than the baselines. One may worry that OneAdapt imposes too much CPU overhead on the sensor that may exceed the sensor-side computation capability. To answer

[9]Note that the bandwidth overhead of streaming DNNGrad from the server to the client is negligible as it contains >7000x less amount of data after sampling and compression of DNNGrad.

this question, we measure the encoding speed on an Intel Xeon 4100 Silver CPU and find that it has enough compute to encode 103.2 video frames per second, which translates to encoding the input data 10 times (as the input data is 10FPS) and is sufficient for OneAdapt in our evaluation setup.

Effectiveness of GPU overhead reduction: OneAdapt introduces two optimizations (§4.1) to reduce the GPU overhead caused by backpropagation: removing unneeded computation and DNNGrad reusing. Figure 11 tests how these two techniques reduce the backpropagation overhead of OneAdapt in terms of GPU runtime and GPU memory on pipeline ⓐ , on a 10-frame video chunk. OneAdapt reduces the GPU runtime overhead of backpropagation by 87% and the GPU memory overhead by 12%. Though removing unneeded computation does not change the DNNGrad, DNNGrad reusing does and may reduce the accuracy or increase the resource usage of OneAdapt. To address this concern, we show that in Figure 12, DNNGrad reusing has negligible impact on the accuracy and resource usage of OneAdapt.

6 RELATED WORK

ML application systems: Almost each ML application, from object detection [29, 32, 42–44, 52, 61–63, 66, 67, 89, 92, 94] to segmentation [42, 43, 66, 67], has seen recent efforts towards efficient systems that with high inference accuracy and reduced resource usage in compute (model inference) and bandwidth/storage (moving data from sensors/sources to the DNN model). They commonly entail various configurations that heavily influence resource usage and/or inference accuracy. For instance, Chameleon [61] incorporates only two video coding parameters (resolution and frame rate), and one parameter that chooses between DNNs. For instance, Elf [95] proposes a new type of knob that partitions the video into different slides and distributes them to a different server for faster inference. Recent work [28, 52, 63] expands system designs with online model training as another configuration, in order to handle the drift of the input content.

Instead of proposing a new system, we design OneAdapt to better adapt a range of existing configurations. That said, OneAdapt as-is does not support for *all* configurations, such as those that modify the analytical DNN model itself.

Adaptation in video analytics systems: To cope with dynamic video input and resource availability, video analytics systems need to timely and optimally set various configurations [28, 42, 43, 61, 81, 87–89, 91, 92]. Two general approaches exist. One relies on (offline or periodic) profiling to search the configuration space for the optimal configuration [89] and leveraging the spatial-temporal locality of the input content to reduce the profiling frequency [61]. However, as elaborated in §2.3, the high overhead of such profiling makes frequent adaptation infeasible. On the other hand, many heuristic-based solutions forgo profiling and instead select configurations based on historical DNN outputs [42, 67, 95] (including the intermediate outputs) or by analyzing the input data using cheap models [29, 43, 66, 81, 92, 94]. This approach adapts quickly, but as shown in §2.3, this approach generally sacrifices optimality and has low accuracy.

Different from prior work, OneAdapt harnesses the differentiability of DNN for configuration adaptation. We use the differentiability of the DNN to cheaply calculate OutputGrad on all knobs with no extra inference and with constant GPU overhead, allowing OneAdapt to frequently adapt to a near-optimal configuration.

7 LIMITATION

Though OneAdapt can optimize a wide range of knobs in streaming media analytics, we have not tested if OneAdapt will work for applications beyond streaming media analytics (such as generative tasks), or knobs that alter the final DNN itself (*e.g.,* DNN selection [61] and DNN customization [28]).

We show that the gradient-ascent strategy of OneAdapt can adapt the configurations when the content of input data changes frequently (*e.g.,* in autonomous driving scenarios). However, this strategy may be sub-optimal when the input content changes too fast (*e.g.,* in car racing scenarios).

When handling more knobs, though the GPU overhead of OneAdapt does not inflate (since OneAdapt only runs one backpropagation), the CPU overhead of OneAdapt still linearly increases for those knobs that the CPU overhead optimization (§4.2) do not apply, constraining the maximum number of knobs that OneAdapt can tune.

Also, the GPU memory overhead of OneAdapt (introduced by running backpropagation) is not negligible. That said, this overhead can be efficiently optimized by gradient checkpointing [31, 51], gradient compression [30, 68] and reversible neural networks [33, 49] and we leave optimizing this overhead to future work.

8 CONCLUSION

OneAdapt addresses a common need of DNN-based applications to timely adapt key configurations over time. The key insight is to harness the differentiability of most DNN models, which allows precise estimation of the gradient of accuracy with respect to all configuration knobs with one backpropagation operation. OneAdapt's improvement (in lower resource usage, or higher accuracy, or both) is validated in four applications, five types of configuration knobs, and five types of streaming media.

ACKNOWLEDGMENTS

We thank the anonymous reviewers and our shepherd Kexin Rong. Junchen Jiang is supported by NSF CNS 2146496, 2131826, 2313190, 1901466, and UChicago CERES Center. The project is also supported by Chameleon Projects [4].

REFERENCES

[1] 1,000-meter autonomous truck perception system - tusimple. https://www.tusimple.com/blogs/tusimple-1000-meter-perception-system/. (Accessed on 09/14/2022).

[2] Audioset. http://research.google.com/audioset/index.html. (Accessed on 02/09/2023).

[3] The best frame rate for video. https://photographylife.com/best-frame-rate-for-video. (Accessed on 09/22/2023).

[4] Chameleon projects. https://www.chameleonprojects.com/. (Accessed on 09/29/2023).

[5] End-to-end deep learning for self-driving cars | nvidia technical blog. https://developer.nvidia.com/blog/deep-learning-self-driving-cars/. (Accessed on 06/09/2023).

[6] Enhancing smart-home experiences with ai-based voice control | electronic design. https://www.electronicdesign.com/technologies/embedded/article/21252996/knowles-electronics-enhancing-smarthome-experiences-with-aibased-voice-control. (Accessed on 09/24/2023).

[7] fairseq/readme.md at main · facebookresearch/fairseq · github. https://github.com/facebookresearch/fairseq/blob/main/examples/wav2vec/README.md. (Accessed on 02/09/2023).

[8] Jpeg - wikipedia. https://en.wikipedia.org/wiki/JPEG. (Accessed on 09/26/2023).

[9] Kuntaidu/oneadapt. https://github.com/KuntaiDu/OneAdapt. (Accessed on 09/29/2023).

[10] lec19.pdf. https://www.cs.princeton.edu/courses/archive/fall13/cos521/lecnotes/lec19.pdf. (Accessed on 06/09/2023).

[11] Microsoft rocket video analytics platform. https://github.com/microsoft/Microsoft-Rocket-Video-Analytics-Platform.

[12] Modified discrete cosine transform - wikipedia. https://en.wikipedia.org/wiki/Modified_discrete_cosine_transform. (Accessed on 09/26/2023).

[13] Mta to install security cameras in nyc subway cars – nbc new york. https://www.nbcnewyork.com/news/local/mta-to-install-security-cameras-in-nyc-subway/-cars-to-deter-crime/3872485/. (Accessed on 09/20/2022).

[14] New york to install surveillance cameras in every subway car. https://www.nbcnews.com/tech/tech-news/new-york-subway-cameras-surveillance-mta-train-/cars-hochul-rcna48582. (Accessed on 09/20/2022).

[15] nsdi17-alipourfard.pdf. https://www.usenix.org/system/files/conference/nsdi17/nsdi17-alipourfard.pdf. (Accessed on 09/24/2023).

[16] N.y.c. subway system to install security cameras in train cars - the new york times. https://www.nytimes.com/2022/09/20/nyregion/nyc-subway-security-cameras.html. (Accessed on 09/20/2022).

[17] OneAdapt: Driving Videos — docs.google.com. https://docs.google.com/spreadsheets/d/1KwRDkt2B7h_WemrK5K86MRz6_Y1czK3bz9u4kBnhvk4. [Accessed 02/15/2023].

[18] Oneadapt proof on numerical gradient case - google docs. https://docs.google.com/document/d/1zSISyfzBirW0kD6hc-lWpj4ykhodobhnPlBQh_T3bBA/edit?usp=sharing. (Accessed on 09/29/2023).

[19] Pytorch_yolov3/yolov3.py at master · dena/pytorch_yolov3 · github. https://github.com/DeNA/PyTorch_YOLOv3/blob/master/models/yolov3.py. (Accessed on 02/09/2023).

[20] Top 10 smart home voice control devices | home matters | ahs. https://www.ahs.com/home-matters/tech/smart-home-voice-control-devices/. (Accessed on 09/24/2023).

[21] Video coding for machines (the moving picture experts group). https://mpeg.chiariglione.org/standards/exploration/video-coding-machines.

[22] Vision navigates obstacles on the road to autonomous vehicles | automate. https://www.automate.org/industry-insights/vision-navigates-obstacles-on-the-road-to-autonomous-vehicles. (Accessed on 06/09/2023).

[23] Voice control in the smart / connected home, what you need to know. https://htacertified.org/app/articles/voice-control-in-the-smart-home/. (Accessed on 09/24/2023).

[24] Waymo datase. https://waymo.com/open/.

[25] x264 ffmpeg options guide - linux encoding. https://sites.google.com/site/linuxencoding/x264-ffmpeg-mapping. (Accessed on 09/10/2022).

[26] Neil Agarwal and Ravi Netravali. Boggart: Towards {General-Purpose} acceleration of retrospective video analytics. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 933–951, 2023.

[27] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in neural information processing systems*, 33:12449–12460, 2020.

[28] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Nikolaos Karianakis, Kevin Hsieh, Paramvir Bahl, and Ion Stoica. Ekya: Continuous learning of video analytics models on edge compute servers. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 119–135, 2022.

[29] Christopher Canel, Thomas Kim, Giulio Zhou, Conglong Li, Hyeontaek Lim, David G Andersen, Michael Kaminsky, and Subramanya R Dulloor. Scaling video analytics on constrained edge nodes. *arXiv preprint arXiv:1905.13536*, 2019.

[30] Ayan Chakrabarti and Benjamin Moseley. Backprop with approximate activations for memory-efficient network training. *Advances in Neural Information Processing Systems*, 32, 2019.

[31] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

[32] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari 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. ACM, 2015.

[33] Vitaliy Chiley, Vithursan Thangarasa, Abhay Gupta, Anshul Samar, Joel Hestness, and Dennis DeCoste. Revbifpn: The fully reversible bidirectional feature pyramid network. *Proceedings of Machine Learning and Systems*, 5, 2023.

[34] Ting-Wu Chin, Ruizhou Ding, and Diana Marculescu. Adascale: Towards real-time video object detection using adaptive scaling. *arXiv preprint arXiv:1902.02910*, 2019.

[35] Vyacheslav V Chistyakov and Panos M Pardalos. Stability analysis in discrete optimization involving generalized addition operations. *Journal of Optimization Theory and Applications*, 167:585–616, 2015.

[36] Lénaïc Chizat. Convergence rates of gradient methods for convex optimization in the space of measures. *arXiv preprint arXiv:2105.08368*, 2021.

[37] Liu Chunhui, Hu Yueyu, Li Yanghao, Song Sijie, and Liu Jiaying. Pku-mmd: A large scale benchmark for continuous multi-modal human action understanding. *arXiv preprint arXiv:1703.07475*, 2017.

[38] High Efficiency Video Coding and ITUT Rec. H. 265 and iso, 2013.

[39] Nando De Freitas, Alex Smola, and Masrour Zoghi. Exponential regret bounds for gaussian process bandits with deterministic observations. *arXiv preprint arXiv:1206.6457*, 2012.

[40] Detectron2. Detectron2 model zoo. https://github.com/facebookresearch/detectron2.

[41] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. {PCC} vivace:{Online-Learning} congestion control. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 343–356, 2018.

[42] Kuntai Du, Ahsan Pervaiz, Xin Yuan, Aakanksha Chowdhery, Qizheng Zhang, Henry Hoffmann, and Junchen 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*, pages 557–570, 2020.

[43] Kuntai Du, Qizheng Zhang, Anton Arapin, Haodong Wang, Zhengxu Xia, and Junchen Jiang. Accmpeg: Optimizing video encoding for accurate video analytics. *Proceedings of Machine Learning and Systems*, 4, 2022.

[44] Kristian Fischer, Felix Fleckenstein, Christian Herglotz, and André Kaup. Saliency-driven versatile video coding for neural object detection. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1505–1509. IEEE, 2021.

[45] Abraham D Flaxman, Adam Tauman Kalai, and H Brendan McMahan. Online convex optimization in the bandit setting: gradient descent without a gradient. *arXiv preprint cs/0408007*, 2004.

[46] Jianqing Gao, Jun Du, and Enhong Chen. Mixed-bandwidth cross-channel speech recognition via joint optimization of dnn-based bandwidth expansion and acoustic modeling. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 27(3):559–571, 2019.

[47] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[48] Alireza Ghasemieh and Rasha Kashef. 3d object detection for autonomous driving: Methods, models, sensors, data, and challenges. *Transportation Engineering*, 8:100115, 2022.

[49] Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. *Advances in neural information processing systems*, 30, 2017.

[50] GoodVision. Goodvision: Smart traffic data analytics. https://goodvisionlive.com/, 2021.

[51] Audrunas Gruslys, Rémi Munos, Ivo Danihelka, Marc Lanctot, and Alex Graves. Memory-efficient backpropagation through time. *Advances in neural information processing systems*, 29, 2016.

[52] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind 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*, pages 123–136. ACM, 2016.

[53] Wei Han, Pooya Khorrami, Tom Le Paine, Prajit Ramachandran, Mohammad Babaeizadeh, Honghui Shi, Jianan Li, Shuicheng Yan, and Thomas S Huang. Seq-nms for video object detection. *arXiv preprint arXiv:1602.08465*, 2016.

[54] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[55] H. G. Hirsch, K. Hellwig, and S. Dobler. Speech recognition at multiple sampling rates. In *Proc. 7th European Conference on Speech Communication and Technology (Eurospeech 2001)*, pages 1837–1840, 2001.

[56] Rattaphon Hokking, Kuntpong Woraratpanya, and Yoshimitsu Kuroki. Speech recognition of different sampling rates using fractal code descriptor. In *2016 13th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pages 1–5, 2016.

[57] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[58] Xuedong Huang, Alex Acero, Hsiao-Wuen Hon, and Raj Reddy. *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Prentice Hall PTR, USA, 1st edition, 2001.

[59] intuVision. intuvision va traffic use case. https://www.intuvisiontech.com/intuvisionVA_solutions/intuvisionVA_traffic, 2021.

[60] Parvaneh Janbakhshi and Ina Kodrasi. Experimental investigation on stft phase representations for deep learning-based dysarthric speech detection, 2021.

[61] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. Chameleon: scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 253–266, 2018.

[62] Daniel Kang, Peter Bailis, and Matei Zaharia. Blazeit: optimizing declarative aggregation and limit queries for neural network-based video analytics. *arXiv preprint arXiv:1805.01046*, 2018.

[63] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. Noscope: optimizing neural network queries over video at scale. *Proceedings of the VLDB Endowment*, 10(11):1586–1597, 2017.

[64] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *Acm Sigplan Notices*, 52(4):615–629, 2017.

[65] Kenji Kawaguchi, Leslie P Kaelbling, and Tomás Lozano-Pérez. Bayesian optimization with exponential convergence. *Advances in neural information processing systems*, 28, 2015.

[66] Yuanqi Li, Arthi Padmanabhan, Pengzhan Zhao, Yufei Wang, Guoqing Harry Xu, and Ravi 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*, pages 359–376, 2020.

[67] Luyang Liu, Hongyu Li, and Marco Gruteser. Edge assisted real-time object detection for mobile augmented reality. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–16, 2019.

[68] Xiaoxuan Liu, Lianmin Zheng, Dequan Wang, Yukuo Cen, Weize Chen, Xu Han, Jianfei Chen, Zhiyuan Liu, Jie Tang, Joey Gonzalez, et al. Gact: Activation compressed training for generic network architectures. In *International Conference on Machine Learning*, pages 14139–14152. PMLR, 2022.

[69] Microsoft. Traffic video analytics – case study report. https://www.microsoft.com/en-us/research/publication/traffic-video-analytics-case-study-report/, 2019.

[70] Arun Narayanan, Ananya Misra, Khe Chai Sim, Golan Pundak, Anshuman Tripathi, Mohamed Elfeky, Parisa Haghani, Trevor Strohman, and Michiel Bacchiani. Toward domain-invariant speech recognition via large scale training. In *2018 IEEE Spoken Language Technology Workshop (SLT)*, pages 441–447, 2018.

[71] Yu Nesterov. Gradient methods for minimizing composite functions. *Mathematical programming*, 140(1):125–161, 2013.

[72] Alexander Neubeck and Luc Van Gool. Efficient non-maximum suppression. In *18th international conference on pattern recognition (ICPR'06)*, volume 3, pages 850–855. IEEE, 2006.

[73] AB Ramazanov. On stability of the gradient algorithm in convex discrete optimisation problems and related questions. 2011.

[74] Xukan Ran, Haolianz Chen, Xiaodan Zhu, Zhenming Liu, and Jiasi Chen. Deepdecision: A mobile deep learning framework for edge video analytics. In *IEEE INFOCOM 2018-IEEE Conference on Computer*

Communications, pages 1421–1429. IEEE, 2018.

[75] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[76] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[77] Mark Schmidt, Nicolas Roux, and Francis Bach. Convergence rates of inexact proximal-gradient methods for convex optimization. *Advances in neural information processing systems*, 24, 2011.

[78] TrafficTechnologyToday. Ai traffic video analytics platform being developed. https://www.traffictechnologytoday.com/news/traffic-management/ai-traffic-video-analytics-platform-being-developed.html, 2019.

[79] TrafficVision. Trafficvision: Traffic intelligence from video. http://www.trafficvision.com/, 2021.

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

[81] Haodong Wang, Kuntai Du, and Junchen Jiang. Minimizing packet retransmission for real-time video analytics. In *Proceedings of the 13th Symposium on Cloud Computing*, pages 340–347, 2022.

[82] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the h.264/avc video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, 2003.

[83] Zhengxu Xia, Yajie Zhou, Francis Y Yan, and Junchen Jiang. Genet: automatic curriculum generation for learning adaptation in networking. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 397–413, 2022.

[84] Xuedou Xiao, Juecheng Zhang, Wei Wang, Jianhua He, and Qian Zhang. Dnn-driven compressive offloading for edge-assisted semantic video segmentation. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pages 1888–1897. IEEE, 2022.

[85] Yi Xiao, Felipe Codevilla, Akhil Gurram, Onay Urfalioglu, and Antonio M López. Multimodal end-to-end autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 23(1):537–547, 2020.

[86] Zhujun Xiao, Zhengxu Xia, Haitao Zheng, Ben Y Zhao, and Junchen Jiang. Towards performance clarity of edge video analytics. In *2021 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 148–164. IEEE, 2021.

[87] Ran Xu, Jayoung Lee, Pengcheng Wang, Saurabh Bagchi, Yin Li, and Somali Chaterji. Litereconfig: cost and content aware reconfiguration of video object detection systems for mobile gpus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 334–351, 2022.

[88] Tiantu Xu, Luis Materon Botelho, and Felix Xiaozhu Lin. Vstore: A data store for analytics on large videos. In *Proceedings of the Fourteenth EuroSys Conference 2019*, pages 1–17, 2019.

[89] Ben Zhang, Xin Jin, Sylvia Ratnasamy, John Wawrzynek, and Edward A Lee. Awstream: Adaptive wide-area streaming analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 236–252. ACM, 2018.

[90] Haixiang Zhang, Zeyu Zheng, and Javad Lavaei. Gradient-based algorithms for convex discrete optimization via simulation. *Operations Research*, 2022.

[91] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael 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, 2017.

[92] Miao Zhang, Fangxin Wang, and Jiangchuan Liu. Casva: Configuration-adaptive streaming for live video analytics. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pages 2168–2177. IEEE, 2022.

[93] Qizheng Zhang, Kuntai Du, Neil Agarwal, Ravi Netravali, and Junchen Jiang. Understanding the potential of server-driven edge video analytics. In *Proceedings of the 23rd Annual International Workshop on Mobile Computing Systems and Applications*, page 8–14, 2022.

[94] Tan Zhang, Aakanksha Chowdhery, Paramvir Victor Bahl, Kyle Jamieson, and Suman Banerjee. The design and implementation of a wireless video surveillance system. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pages 426–438. ACM, 2015.

[95] Wuyang Zhang, Zhezhi He, Luyang Liu, Zhenhua Jia, Yunxin Liu, Marco Gruteser, Dipankar Raychaudhuri, and Yanyong Zhang. Elf: accelerate high-resolution mobile deep vision with content-aware parallel offloading. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, pages 201–214, 2021.

Notation	Definition	Example
n	Number of knobs	$n = 2$
t	t^{th} adaptation interval (by default, each interval is 1 second)	$t = 1$
T	Total number of adaptation intervals	$T = 60$
$\mathbf{k}_t$	Configuration: a vector of knobs with their selected values at interval t	$\mathbf{k}_1$ =(frame rate=10, resolution=480p)
$k_{i,t}$	The value of i^{th} knob in configuration $\mathbf{k}_t$	$k_{2,1}$=480p
Δk_i	A small increase on the i^{th} knob	Δk_2=120p
$\mathbf{x}_t$	input data at interval t	
$\mathbf{r}(\mathbf{k}_t;\mathbf{x}_t)$ or $\mathbf{r}(\mathbf{k}_t)$	Resource usage of config $\mathbf{k}_t$ under $\mathbf{x}_t$. We omit $\mathbf{x}_t$ for simplicity	$\mathbf{r}(\mathbf{k}_1)$ =5Mbps

$Res(\mathbf{k}_t;\mathbf{x}_t)$ or $Res(\mathbf{k}_t)$	Inference results of config $\mathbf{k}_t$ under $\mathbf{x}_t$. We omit $\mathbf{x}_t$ for simplicity.	$Res(\mathbf{k}_1)$ = {
		(obj1, "car", score: 0.2),
		(obj2, "bike", score: 0.8)
		}

e	An element in the inference results (*e.g.,* a detected object). Each element is associated with a confidence score.	e = (obj1, "car", score: 0.2)
θ	Confidence threshold	$\theta = 0.5$ (default)
$\mathbf{y}(\mathbf{k}_t;\mathbf{x}_t)$ or $\mathbf{y}(\mathbf{k}_t)$	DNN input generated by configuration $\mathbf{k}_t$ using input data $\mathbf{x}_t$. We omit $\mathbf{x}_t$ for simplicity.	
$\mathbf{z}(\mathbf{k}_t;\mathbf{x}_t)$ or $\mathbf{z}(\mathbf{k}_t)$	Output utility: number of above-confidence-threshold elements. We omit $\mathbf{x}_t$ for simplicity.	$\mathbf{z}(\mathbf{k}_1) = 1$
$Acc(\mathbf{k}_t;x_t)$ or $Acc(\mathbf{k}_t)$	Accuracy of the configuration $\mathbf{k}_t$ under input data $\mathbf{x}_t$, defined as the similarity between the current inference result $Res(\mathbf{k}_t)$ and the inference results generated using the most resource-demanding configuration.	$Acc(\mathbf{k}_1) = 100\%$
α	Learning rate of OneAdapt	$\alpha = 0.5$ (default)
λ	The hyperparameter that trade-off between accuracy and resource usage.	$\lambda = 1$

Table 4: *Summary of the notations used in OneAdapt*

A NOTATION SUMMARIZATION

Table 4 summarizes the notation used in our paper.

B IMPLEMENTATION OF CONVOLUTION

Beyond the existing GPU optimization, we also provide another optimization that further reduces the GPU computation and memory cost of backpropagation. We contrast the implementation of the convolution operator between normal backpropagation and the backpropagation in OneAdapt in Algorithm 1 and Algorithm 2.

Algorithm 1 Convolution

```
function FORWARD(cache, input, kernel)
    output ← convolution(input, kernel)
    cache.push(input, kernel)
end function
function BACKWARD(cache, outputGrad)
    input, kernel ← cache.pop()
    inputGrad ← convGradInput(gradOutput, kernel)
    kernelGrad ← convGradKernel(outputGrad, input)
end function
```

Algorithm 2 Optimized Convolution

```
function FORWARD(cache, input, kernel)
    output ← convolution(input, kernel)
    cache.push(kernel.abs().mean(dim='channel'))
end function
function BACKWARD(cache, outputGrad)
    kernel ← cache.pop()
    inputGrad ← convGradInput(outputGrad, kernel)
    kernelGrad ← convGradKernel(outputGrad, input)
end function
```

The intuition is that OneAdapt focuses on the DNNGrad of different spatial areas instead of different channels between different channels in the DNN input (*e.g.,* the RGB color channel). We empirically find that this optimization has little impact on the bandwidth–accuracy trade-off of OneAdapt.

C REUSING DNNGRAD

The intuition of reusing DNNGrad is that: although the exact value of DNNGrad may vary across different frames, the spatial areas that have high DNNGrad tend to be stable within several consecutive frames. To verify this intuition, we observe how the similarity (we use cosine similarity) between the saliency of two frames changes with respect to the distance between these two frames (measured by the absolute difference of the video frame id) on a three-second video in our dataset. As shown in Figure 13, we observe that this similarity decreases when the frame distance becomes larger, but the similarity is still greater than 80% when the frame distance is less than 10.

Dataset	Streaming media type	#videos/ #audios	Total length	Description
Traffic [17]	RGB	5	5min	Traffic camera footage
Downtown [17]		10	20min	Driving in downtwon
Country [17]		8	18min	Driving in countryside
PKU-MMD [37]	RGB	10	20min	Human moving in a static scene
	Depth	10	20min	
	Infrared	10	20min	
KITTI [47]	LiDAR	6	4min	Driving on city streets
Google AudioSet [2]	Audio	200	33min	Advertisement and leisure activities

Table 5: *Summary of our dataset.*

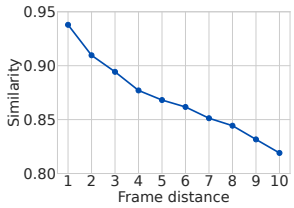


Figure 13: *The similarity of the saliency remains higher than 80% when the frame distance is less than 10.*

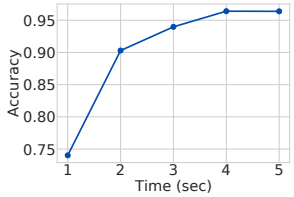


Figure 14: *Empirically OneAdapt converges within 3-5 seconds.*

D SUMMARIZING THE DATASET

Table 5 summarizes the dataset used in OneAdapt.

E THEORETICAL CORRELATION BETWEEN OUTPUTGRAD AND ACCGRAD

In this section, we prove the correlation between AccGrad and OutputGrad by treating them as the analytical gradient (so the chain rule of derivative holds). We are actively working on refining the proof to extend it to numerical gradient case [18].

Definitions: For a given input, we let $Res(\mathbf{k})$ denote the output under a configuration $\mathbf{k}$. We define its accuracy as

$$Acc(\mathbf{k}) = \sum_{e \in Res(\mathbf{k})} f(e)C(e), \tag{3}$$

and the output utility as

$$\mathbf{z}(\mathbf{k}) = \sum_{e \in Res(\mathbf{k})} f(e). \tag{4}$$

where $f(e)$ is a differentiable function that returns a value close to 1 when e is confidently detected (*i.e.,* the confidence

score of e is greater than the confidence threshold θ) and close to -1, otherwise[10], and $C(e) = 1$ if e is confidently detected under the most expensive configuration, and $C(e) = -1$ otherwise.

Theorem: For any knob k in configuration $\mathbf{k}$, we have

$$\underbrace{\frac{\partial Acc(\mathbf{k})}{\partial k}}_{\text{AccGrad}} = \underbrace{\left|\frac{\partial \mathbf{z}(\mathbf{k})}{\partial k}\right|}_{\text{OutputGrad}}.$$

under the following assumptions:

1. For each k in the configuration $\mathbf{k}$ and any element e in $Res(\mathbf{k})$, we have:

$$\frac{\partial f(e)C(e)}{\partial k} \geq 0. \tag{5}$$

2. For any element e in $Res(\mathbf{k})$, there exists a binary matrix $M(e)$ s.t.

$$M(e) \times \frac{\partial \mathbf{z}(\mathbf{k})}{\partial \mathbf{y}} = M(e) \times \frac{\partial \sum_{e' \in Res(\mathbf{k})} f(e')}{\partial \mathbf{y}} = \frac{\partial f(e)}{\partial \mathbf{y}}, \tag{6}$$

where $\times$ means element-wise multiplication and $\mathbf{y}$ refers to DNN input.

3. Finally, we assume

$$\left|\frac{\partial \mathbf{z}(\mathbf{k})}{\partial \mathbf{y}} \otimes \frac{\partial \mathbf{y}}{\partial k}\right| = \left|\frac{\partial \mathbf{z}(\mathbf{k})}{\partial \mathbf{y}}\right| \otimes \left|\frac{\partial \mathbf{y}}{\partial k}\right| \tag{7}$$

Proof:

$$\begin{aligned}
&\frac{\partial Acc(\mathbf{k})}{\partial k} \\
&= \frac{\partial \sum_{e \in Res(k)} f(e)C(e)}{\partial k} && \text{(definition of accuracy)} \\
&= \sum_{e \in Res(k)} \frac{\partial f(e)C(e)}{\partial k} && \text{(linearity of derivative)} \\
&= \sum_{e \in Res(k)} \left|\frac{\partial f(e)C(e)}{\partial k}\right| && \text{(equation 5)} \\
&= \sum_{e \in Res(k)} \left|\frac{\partial f(e)}{\partial k}\right| && (|C(e)| = 1) \\
&= \sum_{e \in Res(k)} \left|\frac{\partial f(e)}{\partial \mathbf{y}} \otimes \frac{\partial \mathbf{y}}{\partial k}\right| && \text{(chain rule,}\mathbf{y}\text{ is DNN input)} \\
&= \sum_{e \in Res(k)} \left|M(e) \times \frac{\partial \mathbf{z}(\mathbf{k})}{\partial \mathbf{y}} \otimes \frac{\partial \mathbf{y}}{\partial k}\right| && \text{(equation 6)} \\
&= \sum_{e \in Res(k)} M(e) \times \left|\frac{\partial \mathbf{z}(\mathbf{k})}{\partial \mathbf{y}} \otimes \frac{\partial \mathbf{y}}{\partial k}\right| && (M(e)\text{is binary matrix)} \\
&= \sum_{e \in Res(k)} M(e) \times \left|\frac{\partial \mathbf{z}(\mathbf{k})}{\partial \mathbf{y}}\right| \otimes \left|\frac{\partial \mathbf{y}}{\partial k}\right| && \text{(equation 7)} \\
&= \left(\sum_{e \in Res(k)} M(e) \times \left|\frac{\partial \mathbf{z}(\mathbf{k})}{\partial \mathbf{y}}\right|\right) \otimes \left|\frac{\partial \mathbf{y}}{\partial k}\right| && \text{(linearity of inner product)} \\
&= \left(\left(\sum_{e \in Res(k)} M(e)\right) \times \left|\frac{\partial \mathbf{z}(\mathbf{k})}{\partial \mathbf{y}}\right|\right) \otimes \left|\frac{\partial \mathbf{y}}{\partial k}\right| && \text{(distributivity)} \\
&= \left|\left(\sum_{e \in Res(k)} M(e)\right) \times \frac{\partial \mathbf{z}(\mathbf{k})}{\partial \mathbf{y}}\right| \otimes \left|\frac{\partial \mathbf{y}}{\partial k}\right| && (M(e) \text{ is non-negative}) \\
&= \left|\sum_{e \in Res(k)} M(e) \times \frac{\partial \mathbf{z}(\mathbf{k})}{\partial \mathbf{y}}\right| \otimes \left|\frac{\partial \mathbf{y}}{\partial k}\right| && \text{(distributivity)} \\
&= \left|\sum_{e \in Res(k)} \frac{\partial f(e)}{\partial \mathbf{y}}\right| \otimes \left|\frac{\partial \mathbf{y}}{\partial k}\right| && \text{(equation 6)} \\
&= \left|\frac{\partial \sum_{e \in Res(k)} f(e)}{\partial \mathbf{y}}\right| \otimes \left|\frac{\partial \mathbf{y}}{\partial k}\right| && \text{(linearity of derivatives)} \\
&= \left|\frac{\partial \mathbf{z}(\mathbf{k})}{\partial \mathbf{y}}\right| \otimes \left|\frac{\partial \mathbf{y}}{\partial k}\right| && \text{(definition of } \mathbf{z}\,(\mathbf{k})\text{, we estimates AccGrad this way)} \\
&= \left|\frac{\partial \mathbf{z}(\mathbf{k})}{\partial \mathbf{y}} \otimes \frac{\partial \mathbf{y}}{\partial k}\right| && \text{(equation 7)} \\
&= \left|\frac{\partial \mathbf{z}(\mathbf{k})}{\partial k}\right| && \text{(chain rule)}
\end{aligned}$$

Empirically, we show in Figure 6, the cosine similarity between AccGrad and OutputGrad still remains over 0.91, indicating that they still have high correlation.

[10]Note that in our evaluation we use $f(e) = 2Sigmoid\,(20(e.score - \theta)) - 1$ (where $e.score$ is the confidence score of e and θ is the confidence threshold), but the choice of f does not affect the validity of our proof.