Chameleon: Scalable Adaptation of Video Analytics

Junchen Jiang[†°], Ganesh Ananthanarayanan[°], Peter Bodik[°], Siddhartha Sen[°], Ion Stoica[⋆]

[†]University of Chicago [°]Microsoft Research [⋆]UC Berkeley, Databricks Inc.

ABSTRACT

Applying deep convolutional neural networks (NN) to video data at scale poses a substantial systems challenge, as improving inference accuracy often requires a prohibitive cost in computational resources. While it is promising to balance resource and accuracy by selecting a suitable NN configuration (*e.g.,* the resolution and frame rate of the input video), one must also address the significant *dynamics* of the NN configuration's impact on video analytics accuracy. We present Chameleon, a controller that dynamically picks the best configurations for existing NN-based video analytics pipelines. The key challenge in Chameleon is that in theory, adapting configurations frequently can reduce resource consumption with little degradation in accuracy, but searching a large space of configurations periodically incurs an overwhelming resource overhead that negates the gains of adaptation. The insight behind Chameleon is that the underlying characteristics (*e.g.,* the velocity and sizes of objects) that affect the best configuration have enough *temporal and spatial correlation* to allow the search cost to be amortized over time and across multiple video feeds. For example, using the video feeds of five traffic cameras, we demonstrate that compared to a baseline that picks a single optimal configuration offline, Chameleon can achieve 20-50% higher accuracy with the same amount of resources, or achieve the same accuracy with only 30-50% of the resources (a 2-3× speedup).

CCS CONCEPTS

• **Information systems** → *Data analytics*; • **Computing methodologies** → *Object detection*;

KEYWORDS

video analytics, deep neural networks, object detection

SIGCOMM '18, August 20–25, 2018, Budapest, Hungary

ACM ISBN 978-1-4503-5567-4/18/08…$15.00
https://doi.org/10.1145/3230543.3230574

1 INTRODUCTION

Many enterprises and cities (*e.g.,* [2, 6]) are deploying thousands of cameras and are starting to use video analytics for a variety of 24×7 applications, including traffic control, security monitoring, and factory floor monitoring. The video analytics are based on classical computer vision techniques as well as deep neural networks (NN). This trend is fueled by the recent advances in computer vision (*e.g.,* [17, 18]) which have led to a continuous stream of increasingly accurate models for object detection and classification.

A typical video analytics application consists of a *pipeline* of video processing modules. For example, the pipeline of a traffic application that counts vehicles consists of a decoder, followed by a component to re-size and sample frames, and an object detector. The pipeline has several "knobs" such as frame resolution, frame sampling rate, and detector model (e.g., Yolo, VGG or AlexNet). We refer to a particular combinations of knob values as a *configuration*.

The choice of configuration impacts both the *resource consumption* and *accuracy* of the video application. For example, using high frame resolutions (*e.g.,* 1080p) or NN models with many layers enables accurate detection of objects but also demands more GPU processing. The "best" configuration is the one with the lowest resource demand whose accuracy is over a desired threshold. Accuracy thresholds are set by the applications, *e.g.,* traffic light changes can function with moderate accuracy while amber alert detection requires very high accuracy. Configurations that meet the accuracy threshold can often vary by *many orders of magnitude* in their resource demands [16, 32], and picking the cheapest among them can significantly impact computation cost.

The best configuration for a video analytics pipeline also *varies over time*, often at a timescale of minutes or even seconds. For the traffic pipeline described above, we may use a low frame-rate (*e.g.,* 5 frames/sec instead of 30 fps) when cars are moving slowly, say at a traffic stop, consuming 6× fewer resources without impacting the accuracy of the vehicle count. In contrast, using a low frame-rate to count fast-moving cars will significantly hurt the accuracy.

As such, we need to frequently change the configuration of the pipeline to minimize the resource usage while achieving the desired accuracy. While prior video analytics systems [16, 32, 33] profile the processing pipeline to minimize cost, they only do so *once*, at the beginning of the video. As a result, these systems fail to keep up with the intrinsic dynamics of the resource-accuracy tradeoff, and they end up either

wasting resources (by picking an expensive configuration) or not meeting the accuracy target.

One natural approach to address this challenge is to *periodically profile* the pipeline configurations to find an optimal resource-accuracy tradeoff. Unfortunately, this is prohibitively expensive because the number of possible configurations is exponential in the number of knobs and their values. Even a simple video pipeline with just a few knobs can have thousands of potential configurations. Further, the cost of executing some of the configurations can be orders of magnitude higher than the most efficient one we end up selecting. In fact, in our early experiments, the cost of periodic profiling often exceeded any resource savings gained by adapting the configurations. The main challenge, thus, is to significantly *reduce the resource cost of periodic configuration profiling*.

Unfortunately, using traditional modeling techniques such as Bayesian optimization [12], multi-armed bandits [19], or optimal experiment design [31] to update pipeline configurations at the granularity of seconds is very expensive, due to the number of required experiments. In fact, these techniques typically assume a stationary environment, where it is sufficient to profile once upfront or infrequently (once a day). Our setting, however, is *non-stationary*. For instance, tracking vehicles when traffic moves quickly requires a much higher frame rate than when traffic moves slowly, but when each condition occurs may vary by hour, minute, or second.

To address this challenge, we take a more direct approach that leverages domain-specific insights on the *temporal and spatial* correlations of these configurations.

Temporal correlation: While the best configuration varies over time, certain characteristics tend to persist. The top-k best configurations (cheapest k configurations with accuracy above the desired threshold) tend to be relatively stable over time, for a small value of k. Similarly, configurations that are very bad—very inaccurate and/or very expensive—remain so over long time periods. Thus we can significantly prune the search space during profiling by learning which configurations are promising and which are unhelpful.

Cross-camera correlation: Video feeds of cameras deployed in geographical proximity (*e.g.,* in the same city or building) often share properties (*e.g.,* the velocities and sizes of objects) that affect the optimal configuration. Instead of searching for optimal configurations per camera feed, we can *amortize the profiling cost across multiple cameras*. Once we identify a good set of configurations for one video feed, we can reuse it on similar feeds. As more organizations deploy large fleets of cameras, leveraging cross-camera correlations will become an increasingly effective way to reduce the cost of profiling.

Independence of configurations: A central question in any supervised learning problem is how to obtain high-quality labels. One possibility is to use humans to label the video feeds frame by frame, but this approach is slow and unscalable. Instead, we use an expensive *golden configuration* to provide the "ground truth", following prior work [22, 24, 32]. An example of such a configuration for a traffic pipeline is a resolution of 1080p, at 30 fps, using a full Yolo model. However, to minimize the cost of running the golden configuration, which uses the best (and most expensive) values for all knobs, we rely on an empirical observation that knobs are typically *independent*. That is, for a given configuration knob, the relationship between its resource and accuracy is largely independent of the values of the other configuration knobs. Thus, in our profiler, we measure the value of a given knob (*e.g.,* frame rate) by simply holding the values of the other knobs fixed (*e.g.,* to a reasonably inexpensive value).

In this paper, we leverage these observations to develop *Chameleon*, a video analytics system that optimizes resource consumption and inference accuracy of video analytics pipelines, by adapting their configurations in real-time. Using live video feeds from five real traffic cameras, we show that compared to a baseline that picks the optimal configuration offline, Chameleon can achieve 20-50% higher accuracy with the same amount of resources, or achieve the same accuracy with only 30-50% of the resources (2-3× speedup).

Our key contributions are as follows:

- We do a cost-benefit analysis for continuously adapting NN configurations compared to one-time tuning, and show that it can save compute resources by up to 10× and raise accuracy by up to 2×. (§3)
- We identify and quantify the impact of spatial and temporal correlations on resource-accuracy tradeoffs. (§4)
- We present a suite of techniques to dramatically reduce the cost of periodic profiling by leveraging the spatial/temporal correlations. (§5)

2 RESOURCE-ACCURACY PROFILES

We begin with some background on NN-based video analytics, and use the object detection pipeline to show the impact of configurations on inference accuracy and cost.

2.1 Object detection pipelines

We use object detection as a running example to illustrate our ideas. The goal of object detection is to identify objects of interests, their classes (*e.g.,* car), and sometimes their locations in each frame of the video. Object detection is a core vision task on which a wide range of higher-level tasks are built, so improving it can impact many applications.

The simplest way to detect objects in a live video stream today is to decode each frame into a bitmap and run each bitmap through an object-detection NN, such as Yolo [9] or Faster RCNN [15]. This would detect all objects in all the frames, but would require a significant amount of GPU resources for NN inference. Such an expensive approach may be necessary if we want to detect even small objects and the objects present change very frequently. However, in many scenarios, the objects might change very slowly (*e.g.,* each object stays on screen for at least a second) and

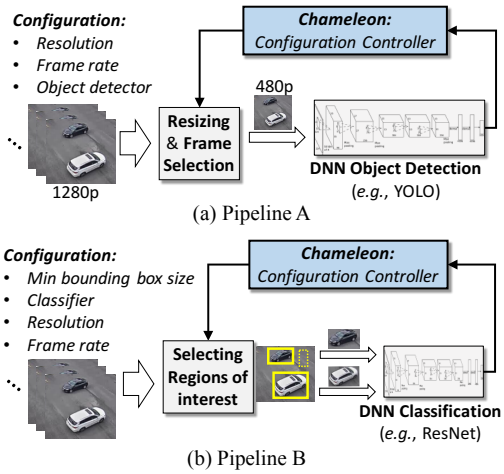


Figure 1: *Two typical NN-based object detection pipelines and their configuration knobs. Pipeline A uses a single NN to detect and classify objects in a given frame, while Pipeline B has two separate steps to detect regions of interest as bounding boxes and then classify each bounding box using an NN classifier, which is often cheaper than using the object detection NN. Thus, Pipeline A is more generally applicable, but Pipeline B is preferred if the regions of interest are few and easy to identify (e.g., moving objects in front of a static background).*

we may only want to detect relatively large objects. In this case, processing only 1 frame per second and resizing it from 960p to 480p, for example, would reduce resource demand by 120× with essentially no impact on accuracy. Frame sampling and resizing are just two of many possible knobs in a video processing pipeline that can dramatically reduce resource demand with only a small impact on accuracy.

Pipelines: We consider two object detection pipelines, as shown in Figure 1. In pipeline *A*, the raw video frames are first pre-processed by sampling frames (to reduce the frame rate) and resizing them, and are then fed into one of several pre-trained object-detection models (*e.g.,* Faster RCNN [15], Yolo [9]). Pipeline *B* uses a light-weight background subtraction logic (a non-NN model for which CPU is sufficient) to first detect regions with motion, and only sends these smaller regions to an NN-based classifier (e.g., ResNet [30], MobileNet [21]) to label them. Both pipelines have been actively studied in the computer vision literature. Although pipeline *A* has attracted more attention recently due to NN advancements, pipeline *B* is often a better, cheaper choice if the camera is static (*e.g.,* mounted on a pole).

Configurations: While logically both pipelines expose similar interfaces, they have different sets of configuration *knobs*, whose values are critical to the performance (accuracy and resource consumption) of object detection. We focus on a different subset of knobs from each pipeline to create two illustrative examples, and use them throughout the paper.

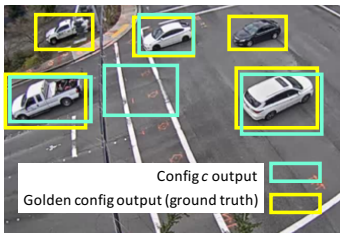


Figure 2: *An illustrative example of the accuracy metric: Precision = 3 (# of true positives) / 5 (# of detected objects), Recall = 3 (# of true positives) / 4 (# of objects), F1 score = 2/(1/Precision+1/Recall) = 2/3.*

- Three knobs from Pipeline *A*: *frame rate* ({30, 10, 5, 2, 1}fps), *image size* ({960, 840, 720, 600, 480}p), and *object detection model* (FasterRCNN+{InceptionResNet, ResNet101, ResNet50, InceptionV2}, SSD+{InceptionV2, MobileNetV1}) [7]. The frame rate and image size decide which frames and in what size they should be fed to the object detection model.
- Two knobs from Pipeline *B*: *minimum size* of the region with detected motion (to ignore spurious detections) as a fraction of the whole frame ({0.5, 0.1, 0.05, 0.01, 0.005}%), and the *classifier model* (ResNet101, ResNet50, InceptionV2, MobileNetV1) [8]. Only regions larger than the minimum size are sent to the classifier for inference.

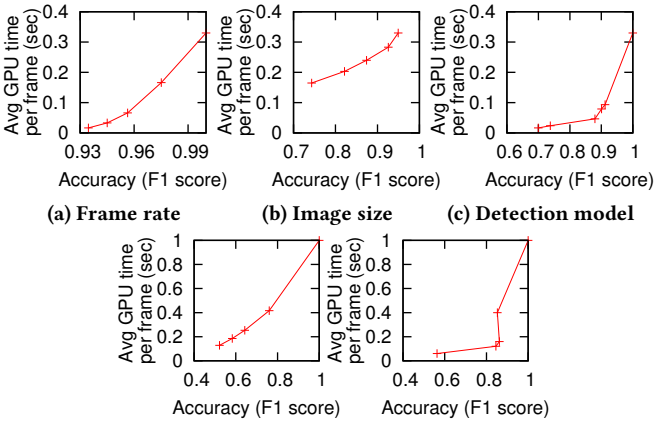
The configuration space comprises all possible combinations of values of these knobs: in total, Pipeline A has 150 configurations and Pipeline B has 20.

The above two pipelines illustrate a few key properties common to video analytics. These pipelines include a mix of NN as well as classic computer vision modules, along with their relevant knobs (*e.g.,* region size for background subtraction). The pipelines also represent a *cascade* of modules, where an upstream module (*e.g.,* frame sampler) decides if a frame will be processed by downstream modules. Thus, using the above two pipelines in our study will help us develop techniques that hold across many video analytics pipelines.

2.2 Performance of configurations

The performance of a configuration on a set of frames is measured by two metrics: accuracy and cost.

Accuracy: When using configuration *c*, we compute accuracy of a single frame by comparing the detected objects with the objects detected by the most expensive configuration, which we call the *golden configuration*, using the F1 score, which is the harmonic mean of precision and recall. Figure 2 shows an illustrative example. We identify true positives in the F1 score using two conditions: (1) a *bounding box-based* condition, which checks if the detected bounding box has the same label as some ground truth box; or (2) a *label-based* condition, which checks if the bounding box has the same label *and* sufficient spatial overlap with some ground truth box [14]. Both metrics are useful in real applications and used in our evaluation (§6), consistent with prior work [24, 32].



(a) Frame rate (b) Image size (c) Detection model

(d) Min bounding box size (e) Classifier

Figure 3: *Impact of each configuration knob on resource-accuracy tradeoffs. The accuracy threshold $\alpha = 0.8$.*

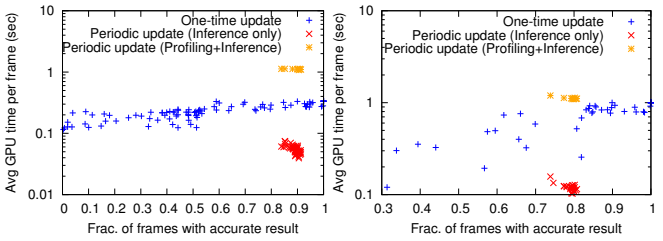
To compute the accuracy of a frame that was not sampled by c, we use the location of objects from the previous sampled frame. For a video segment, which consists of many frames, we compute accuracy as the fraction of frames with F1 score $\geq \alpha$.[1] The above metric for accuracy nicely lends itself to composing application-level metrics. For instance, in a traffic analytics deployment [25], measuring the fraction of frames with F1 score ≥ 0.7 was a good indicator of the error in the application-level traffic counts. Note that our techniques can also directly work with other accuracy metrics.

Cost (resource consumption): We use average GPU processing time (with 100% utilization) per frame as the metric of resource consumption, because GPU is the dominant resource for the majority of video processing workloads. Further, the performance of NN-based inference is more dependent on GPU cycles than typical data analytics tasks.

Performance impact: Figure 3 shows how the configuration knobs affect the performance of object detection, measured by accuracy and resource consumption. We use a dataset of 120 clips of traffic videos (30fps frame rate and 960p resolution, see §6.1 for details). Different points represent the resource-accuracy tradeoffs of setting each knob to different values while fixing other knobs to their most expensive values. We see that one can reduce resource consumption by tuning the values of these configurations, an observation that has informed other work [32].

At the same time, however, we notice that reducing resource consumption leads to substantial accuracy degradation. This is because a fixed configuration is used for the entirety of each video (several minutes), during which the content changes significantly. In the next section, we show that the relationship between configuration and accuracy has great temporal variability, so dynamically adapting the configuration can lead to better resource-accuracy tradeoffs.

[1] Calculating an overall accuracy based on per-frame accuracy of consecutive frames could be biased due to correlations in the frames' content, but we mitigate this by using long videos sampled across different hours (§6.1).



(a) Pipeline A (b) Pipeline B

Figure 4: *Potential benefit of updating the NN configuration periodically (every $T = 4$ seconds). Ignoring profiling, both accuracy and cost significantly improve (red), but when profiling is factored in, the cost is worse (yellow) than one-time profiling.*

3 POTENTIAL OF ADAPTATION

The basic premise of Chameleon is that videos and the characteristics of video analytics pipelines exhibit substantial dynamics over time. As a result, to achieve the "best" resource-accuracy tradeoff, we need to continuously adapt the configurations of the video pipelines.

3.1 Quantifying potential

We first show the value of continuous adaptation by comparing two simple policies for selecting NN configurations.

1. *One-time update:* This is a one-time offline policy that exhaustively profiles all configurations on the first x seconds of the video, picks the cheapest configuration that has at least α accuracy, and sticks with it for the whole duration of the video (*e.g.*, [**?**]). We use $x = 10$.
2. *Periodic update:* This policy divides the video into T-second intervals, and profiles all configurations in each interval for the first t seconds of that interval. It then sticks with the cheapest configuration whose accuracy is greater than α for the rest of the interval, *i.e.*, for $T - t$ seconds. We use $T = 4$ and $t = 1$ for our experiments. We examine how sensitive the results are to T in §6.3.

Figure 4 shows the resource-accuracy tradeoffs of running the two policies on 30 traffic videos (there are 30 dots per color). We set the target accuracy threshold α to 0.7 and 0.8, respectively (we observe similar results with other thresholds). The figures show that the periodic policy (red) reduces per-frame resource consumption by over 10× and improves the accuracy by up to 2× over the one-time policy (blue).

Intuition: The intuition behind these improvements is that the accuracy of a given configuration can depend heavily on the video content. If the video content becomes more challenging (*e.g.*, traffic moves faster, or there is less lighting), using the same configuration will negatively impact the accuracy. Instead, we need to move to another configuration that will increase the accuracy, likely at the expense of using more resources. Similarly, if the video content becomes

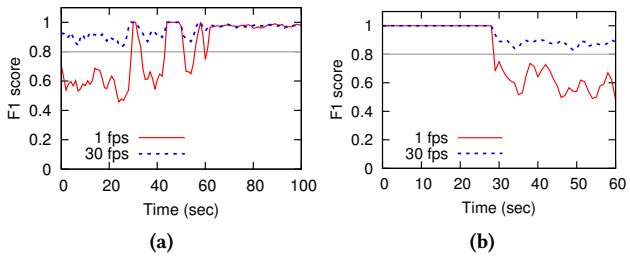


Figure 5: *Time-varying accuracy of two video clips. The grey lines show the accuracy threshold. Picking configurations using the first 20 minutes causes resource waste (5a) or low accuracy (5b) in the rest of the video.*

less demanding, we might have the opportunity to move to another configuration that consumes less resources, while still maintaining the desired accuracy.

Figure 5 illustrate this intuition, by plotting the accuracy of the two policies over time for two video clips. In Figure 5a, initially the cars are moving slowly, so the one-time policy picks a low frame sampling rate that is sufficient to achieve the desired accuracy. After a while, however, the cars start moving much faster, and the low sampling rate is no longer sufficient to correctly detect them. In contrast, the periodic policy is able to maintain high accuracy by increasing the frame sampling rate. Figure 5b shows the converse: cars start out moving very fast, causing both policies to start with a high frame sampling rate. When the cars slow down, the periodic policy reduces the frame sampling rate and saves GPU resources by over 2× compared to the one-time policy. Cases like Figure 5, where accuracy varies significantly over time, are common in real-world camera streams.

3.2 Prohibitive profiling cost

While Figure 4 shows considerable potential of the periodic policy, a big caveat is that these results do not include the profiling cost; they only include the cost of running the selected configuration. Not surprisingly, profiling all configurations every T seconds induces a significant *profiling cost*. Worse yet, this profiling cost grows exponentially in the number of configuration knobs and the number of values per knob.

If done naively, the profiling cost of the periodic policy can negate any gains made by dynamically adapting the configuration. As shown in Figure 4, the total cost when including profiling (yellow) is almost 20× higher than the actual cost of running the selected configuration (red), and it is at least as high as (or in Pipeline A, over 3× higher than) profiling configurations once (blue).

A sizable component of this profiling cost comes from running the golden configuration. Recall from §2.2 that we use the golden configuration to obtain the ground truth for evaluating the accuracy of a given configuration. On average, running the golden configuration requires an order of magnitude more GPU resources than other configurations; *e.g.,* running a pre-trained FasterRCNN+ResNet101 model on a video encoded in 1280p and 30fps requires 10× more GPU resources than running a pre-trained SSD-MobileNet model on the same video encoded in 1280p and 10fps.

3.3 Challenges in reducing profiling cost

At first sight, it might appear that using a state-of-the-art search algorithm (*e.g.,* [12, 19]) could address the prohibitively high profiling cost of the periodic policy. Unfortunately, these algorithms are inadequate for our setting.

First, existing algorithms are designed for offline settings (*e.g.,* find the optimal cloud configuration for a Hadoop application [12]). In contrast, our setting requires *periodic* profiling, where the cost of a profiling event must be significantly lower than the actual cost of running the selected configuration between two profiling events. Second, the profiling cost includes the cost of running the golden configuration, which itself can be prohibitively expensive.

Note that simply increasing the update interval T does not help in practice. If the update interval is too large, we might either miss changes in the video content, which could negatively impact accuracy, or miss opportunities to reduce the resource consumption.

In summary, naive continuous profiling is expensive for three reasons. We have to frequently run the golden configuration on each video stream, we have to profile all video streams, and the configuration space is exponentially large.

4 KEY IDEAS IN CHAMELEON

Chameleon tackles these challenges using three domain-specific empirical observations about the impact of configurations on the accuracy and cost of video analytics pipelines. First, if the NN configurations' resource-accuracy tradeoff is affected by some persistent characteristics of the video, we can learn these *temporal correlations* to reuse configurations over time (§4.1). Second, if two video feeds share similar characteristics, it is likely they will also share the same best configurations. Such *cross-camera correlations* provide an opportunity to amortize profiling cost across multiple camera feeds (§4.2). Finally, we have experimentally observed that many of the configuration knobs *independently* impact accuracy, allowing us to avoid an exponential search (§4.3).

Notice that these empirical observations are not specific to the videos in our dataset, nor to the profiling algorithm we use. The next sections provide the intuition behind the observations and discuss their implications on Chameleon.

4.1 Persistent characteristics over time

While the characteristics of videos change over time, the underlying characteristics of the video objects (*e.g.,* size, class, viewing angle) that affect accuracy tend to remain relatively stable over time. As a result, configurations that are particularly bad tend to remain so. For example, if a camera is covering objects from a side view and an object detector is not tuned to detect objects at a side-view angle, it will almost always produce results with low accuracy and hence

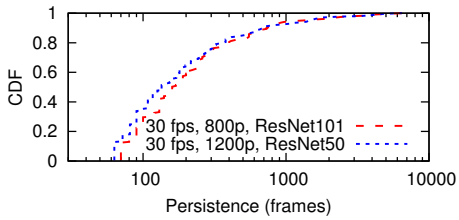


Figure 6: *Distribution of persistence of two configurations, run on 120 150-second video clips.*

should not be selected. Such configurations can be learned and discarded from our profiling for longer periods of time.

Similarly, good configurations also tend to consistently produce good performance. A common example is surveillance video, which tends to have static content. For such videos, a low-cost configuration (*e.g.,* low frame rate) would be sufficient for a long period of time. More generally, even though the best configuration, *i.e.,* the one with the lowest cost meeting the accuracy threshold α, might change frequently, the set of *top-k best configurations* (top-k cheapest configurations with accuracy $\geq \alpha$) tend to remain stable over time. Thus, we can dramatically reduce the search space by focusing on these top-k configurations.

To show the distribution of time intervals over which the video characteristics remain relatively stable, we define the *persistence* of a configuration c as the length of contiguous frames for which c's accuracy *exceeds a threshold α for over 95% of the frames* (we do not use 100% to avoid noise). Figure 6 shows the distribution of the persistence of two typical configurations. We observe that half of the time, the configurations persistently exceed the accuracy threshold for more than 200 frames (roughly 6 seconds at 30fps).

Neither of these rules holds all of the time. Thus, we periodically explore the full configuration space, to give previously bad configurations the opportunity to prove their worth, and allow previously good configurations to fall from grace.

4.2 Cross-camera similarities

Video feeds that exhibit spatial correlations are abundant in practice. For instance, the traffic cameras facing a highway section may be correlated, because same vehicles may appear in the video feeds of multiple cameras.

Even when cameras do not observe the same scene, their video feeds can still have similar characteristics. Figure 7a and 7b show two similar traffic cameras deployed in a city. The vehicles in the city will likely have similar moving speeds, lighting that is influenced by weather conditions, and viewing angles due to the cameras being installed at similar heights (as a result of uniform installation policies). Even if the cameras are not in geographic proximity, cameras deployed for the same purpose such as traffic analytics are likely to exhibit similarities. This can happen, for example, due to the underlying similarity of street planning across a country. Such similarity can also occur for surveillance cameras covering an enterprise building. Figure 7c and 7d

(a) Camera #1 **(b) Camera #2**

(c) Camera X **(d) Camera Y**

Figure 7: *Screenshots of similar traffic video feeds (a, b) and two similar indoor video feeds (c, d).*

show screenshots of two indoor cameras (see §6.1 for details) who do not share field of view but have similar classes of objects (*e.g.,* humans), temporal patterns of movement (*e.g.,* more people movement during lunch/dinner time), lighting, etc., so they tend to share the best configuration.

As more cameras are deployed with increasing spatial density, we expect more opportunities of amortizing the profiling cost over many similar video feeds, especially in real-time applications such as traffic monitoring and building surveillance, where many cameras have similar video feeds.

Note that the spatial correlations of best configurations do not mean that applying the same configuration will produce the exact same accuracy on different videos. The timescales at which such correlations emerge can be larger than the timescale at which the accuracy changes. Thus, we should not blindly reuse the best configuration on another video, even when the characteristics of the two videos are deemed very similar. Instead, we can leverage the fact that similar videos tend to have similar *distributions* of best configurations. Then, we can use the most promising configurations from one camera—*e.g.,* the top-k best configurations—to guide the profiling of a spatially-related camera. Finally, we do not simply apply the same configurations on *all* cameras. For instance, in Figure 8, Camera #1 and #2 (from Figure 7) have similar resource-accuracy profiles that are different from that of Camera #3 (not shown in Figure 7), so the first two cameras can share the set of best configurations, but not with the third camera. We will describe our data-driven approach for automatically grouping similar cameras in §5.3.

4.3 Independence of configuration knobs

To further reduce the cost of searching the exponential configuration space, we observe that, typically, individual knobs have *independent* impact on accuracy. For example, consider a pipeline with the resolution and frame rate knobs, taking values (480p,720p,960p) and (1, 10, 30), respectively. Recall from §3 that we measure the accuracy of configurations relative to a golden configuration, which in this case is (960p, 30). We observe empirically that in most cases the accuracy

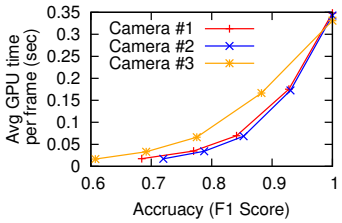


Figure 8: *Spatially similar cameras tend to have similar resource-accuracy profiles. By varying frame rate, Camera #1 and Camera #2 share a similar resource-accuracy profile, which is different to Camera #3.*

of configuration (480p, 30) relative to (960p, 30) *is similar in value to* the accuracy of (480p, 10) relative to (960p, 10).

This has two important implications. First, it lets us tune the resolution knob *independent* of the frame rate; this prunes a large part of the configuration space. Second, it lets us estimate a configuration's accuracy by combining its per-knob accuracies; in particular, we can do this *without* running the expensive golden configuration.

Further, the *relative ordering* between the configurations on their resource demand will be unaltered between using frame rates 30 and 10. That is, if the resource demand of (480p, 30) is less than that of (720p, 30), then this ordering will continue to be true with the resource demand of (480p, 10) being less than (720p, 10). In our setting, the configuration knobs have *monotonic* impact on cost, *i.e.,* increasing the value of a knob while holding the other knobs fixed increases the resource demand of the configuration.

Since our objective is to pick the cheapest configuration that meets the desired accuracy threshold, the above observations allow us to significantly reduce the profiling costs.

5 CHAMELEON TECHNIQUES

We build upon the insights in §4 to present the techniques in our solution, Chameleon, to reduce the cost of profiling for online adaptation of configurations. It should be noted that Chameleon only represents one instance of a concrete design inspired by the insights in §4, which are of independent value.

5.1 Overview

Chameleon's solution relies on periodically re-profiling the video pipeline. Video workloads are typically non-stationary, in that both the characteristics of the video as well as the pipeline tend to change over time. This makes traditional approaches (*e.g.,* Bayesian optimization, multi-armed bandits) either too expensive for real-time adaptation or unsuited because they assume a stationary environment (despite their successful use in recent systems [12, 19, 31]).

Chameleon uses a solution inspired by greedy hill climbing that exploits the independence of NN configuration knobs to reduce the search space from exponential to linear (§5.4). Using this profiling method, it leverages the temporal persistence of configurations to learn their properties over time

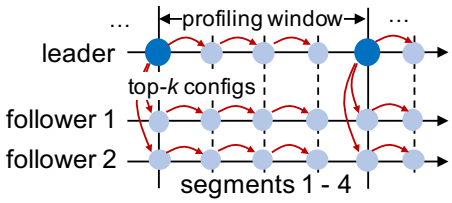


Figure 9: *The horizontal lines represent video feeds generated by three cameras over time (one leader, two followers). The solid vertical lines delineate profiling windows and the dashed lines delineate segments. The blue circles represent profiling (§5.4); bigger dark circles represent full profiling of the configuration space, which yields the top-k most promising configurations from the leader. The red arrows show the propagation of the top-k configs both temporally (to segments on the same camera, §5.2), and spatially (to segments on the follower cameras, §5.3).*

(§5.2), and amortizes the cost of profiling across multiple cameras by leveraging cross-video similarities (§5.3).

Figure 9 illustrates how different Chameleon techniques work together. A "leader" video is profiled at the start of a "profiling window" and a set of good (top-k) configurations is found. This set is shared among "follower" videos who are similar to the leader. Both the leader and followers then restrict their search to this top-k set when choosing configurations over time, until the start of the next profiling window (when a new top-k set will be obtained from the leader). All of these terms are defined in the subsequent sections.

5.2 Temporal incremental updates

Chameleon periodically and profiles the video pipeline using an interval we refer to as the *profiling window*. Each profiling window is split into w smaller *segments*, each of which is a contiguous set of frames spanning a T-second interval (by default, $T = 4$). We leverage temporal persistence (§4.1) by not profiling all the segments in a profiling window. Instead, we re-profile the configuration space on only the first segment of the profiling window (see §5.4 for details). We use the results from this first segment to obtain a short ranked list of the top-k most promising configurations (*i.e.,* the least expensive k configurations that meet the accuracy threshold), and then profile only the top-k configurations on the remaining segments to find the best one. When profiling a configuration on a segment, we do not evaluate every frame of the segment. Instead, we profile the first t seconds (by default, $t = 1$) and use the best configuration for the remaining $T - t$ seconds.

Algorithm 1 lists the steps taken in each profiling window given a video stream i. For the i^{th} video, we use $S_{i,j}$ to denote the j^{th} segment and $W_{i,l}$ to denote the l^{th} profiling window, which is a list of w consecutive segments $(S_{i,wl}, \ldots, S_{i,w(l+1)-1})$. Line 3 in the algorithm picks the top-k configurations from the first segment, and lines 5-7 use the top-k set to profile the remaining segments. Both these steps use the `Profile` method, which we will cover in §5.4.

Input: $W_{i,l}$: the l^{th} profiling window of the i^{th} video, which consist of w segments $S_{i,wl}, \ldots, S_{i,wl+l-1}$; C: set of all configs under consideration
Output: A map from each segment $S_{i,j}$ in the profiling window to the chosen config $\hat{c}_{i,j}$.

1 **Function** `UpdateWindowT`$(W_{i,l}, C)$**:**
2 $\quad Result \leftarrow \emptyset$
3 $\quad C_l^{promising} \leftarrow$ `Profile`$(S_{i,wl}, C, k)$
4 $\quad Result.add(\hat{c}_{i,wl}, C_l^{promising}[0])$
5 $\quad$ **foreach** $j = wl+1, \ldots, wl+w-1$ **do**
6 $\quad\quad \hat{c}_{i,j} \leftarrow$`Profile`$(S_{i,j}, C_l^{promising}, 1)[0]$
7 $\quad\quad Result[s_{i,j}] \leftarrow \hat{c}_{i,j}$
8 $\quad$ **return** *Result*

Algorithm 1: *Temporal updates take promising configurations from the first segment of a profiling window and apply them to subsequent segments in the window.*

We have also observed in practice that it is beneficial to include not only the current set of top-k configurations but also those from the past few profiling windows. So we can sample from $\cup_{l \in \mathbb{W}} C_l^{promising}$ to obtain the k configurations, where $\mathbb{W}$ is the set of profiling windows in the past. This extension is currently not used in our experiments.

5.3 Cross-video inference

The ability to take good configurations profiled on one video stream and apply them to other video streams offers more potential savings (§4.2).

Cross-video profiling: In each profiling window, Chameleon leverages spatial similarities in NN performance to amortize the cost of profiling across multiple video feeds. Let P be the cost of profiling a video segment on the full configuration space (C), $p \ll P$ the cost of profiling only the top-k most promising configurations from a reference video ($C^{promising}$), and suppose there are V videos in total. If the videos need to be profiled separately, the cost of profiling is $P \cdot V$ in every profiling window. On the other hand, if the videos show spatial similarity, the cost reduces to $P + p(V-1)$, a savings that grows with the number of videos V.

Algorithm 2 takes a group of *related* video streams G as input, and only profiles the first video (referred to as the "leader") on the full configuration space (lines 3-5). We discuss how we group the set of related videos shortly. The call to `UpdateWindowT` assigns a configuration to each segment of the window using the temporal update technique from §5.2. For all remaining videos (the "followers"), the call to `UpdateWindowT` only receives the most promising configurations from the first video as input (line 7), thus significantly reducing the search space.

Grouping related videos: We group video feeds by exploiting the correlation between configurations on the accuracy of their output on different feeds; these accuracy values are comparable across feeds since they are running the same pipeline. The grouping of video feeds is offline and done relatively infrequently (*e.g.,* once every few hours).

Input: G: a list of related video feeds, C: set of configs under consideration
Output: Configuration $\hat{c}_{i,j}$ for each segment $S_{i,j}$.

1 **Function** `UpdateWindowS`(l, G, C)**:**
2 $\quad Result \leftarrow \emptyset$
3 $\quad leader \leftarrow G[0]$
4 $\quad Result.add($`UpdateWindowT`$(W_{i,l}, C))$
5 $\quad C_l^{promising} \leftarrow$ (returned by `Profile` in line 4)
6 $\quad$ **foreach** $i \in G \setminus \{leader\}$ **do**
7 $\quad\quad Result.add($`UpdateWindowT`$(W_{i,l}, C_l^{promising}))$
8 $\quad$ **return** *Result*

Algorithm 2: *Spatial updates take promising configurations profiled from one video and apply them to all other videos in the same related group.*

We use the following simple grouping algorithm. The algorithm starts with a randomly chosen configuration and profiles all videos on it, then uses the accuracy results to create an initial grouping using a simplified version of k-means—essentially, sort the accuracy values and bound the deviation from the minimum of a group. We repeat this process by using another randomly chosen configuration to subdivide the groups created by the previous round, and so on. This greedy refinement stops when enough configurations have been tested or the groups become too small.

While the above grouping algorithm works well in our evaluations, we realize that it represents a very simple, first stab at the problem, and there could be value in adding more sophistication. For example, the algorithm could be augmented with information on camera specifications and object density. We defer these and other improvements to future work, while currently choosing to focus on the feasibility and potential savings of grouping videos.

5.4 Profiling a video segment

Algorithm 1 reduces the profiling cost from once per video segment per video feed to once per profiling window per video feed. Algorithm 2 further reduces the profiling cost from once per profiling window per video feed to once per profiling window per video group. Although these savings are substantial, even profiling once can be very costly. This is because the configuration space is multi-dimensional, consisting of several knobs each taking one of several values, yielding exponentially many possible configurations. Assuming m knobs and (for simplicity) n values for each knob, an exhaustive search would involve $O(n^m)$ configurations. Instead, Chameleon leverages the empirically-driven assumption from §4.3 that the knobs of our NN configurations can be treated independently. This allow us to use a variant of

Input: S: a segment; C: set of configs under consideration, k: # of top configs to output.
Output: A list of the top-k best configurations in descending order.

1 **Function** `Profile`(S, C, k)**:**
2 $KnobValueToScore \leftarrow \emptyset$
`/* Profile one knob at a time */`
3 **foreach** *Knob* r **do**
4 **foreach** $v_r \in V_r$ **do**
`/* Set knob` r `to` v_r `while setting others to the golden value` v_r^* `*/`
5 $c(v_r) \leftarrow (v_1^*, \ldots, v_r, \ldots, v_n^*)$
`/*` c^* `is Golden config */`
6 $c^* \leftarrow (v_1^*, \ldots, v_r^*, \ldots, v_n^*)$
`/* Calculate F1 metric of` $c(v_r)$ `by comparing` $c(v_r)$ `against` c^* `*/`
7 $f \leftarrow Eval\_F1(S, c(v_r), c^*))$
8 $KnobValueToScore[v_r] \leftarrow f$
9 $AccurateConfigs \leftarrow \{c \in C | \prod_r KnobValueToScore[v_r] \geq \alpha'\}$
10 Sort $AccurateConfigs$ by c's resource consumption
11 $AccurateConfigs \leftarrow$ top k elements in $AccurateConfigs$
12 **return** $AccurateConfigs$

Algorithm 3: *Finds the top-k best configurations for a segment efficiently by profiling each knob independently (a form of greedy hill climbing).*

greedy hill climbing, where each knob is tuned while all other knobs are held fixed, reducing the search space to O(mn).

Algorithm 3 shows how Chameleon profiles each knob independently. For each value v_r of knob r, we construct a configuration $c(v_r)$ with knob r set to v_r while all other knobs are set to their golden configuration (maximum) values. We compare $c(v_r)$ to the golden configuration c^*, which sets r to its most expensive value (lines 5-7). $Eval\_F1$ computes the average F1 score of a configuration c with respect to another configuration c' over the frames in S, *i.e.,* $\frac{1}{|S|}\sum_{s \in S} F1(s, c, c')$. The final accuracy of a configuration $c = (v_1, \ldots, v_n)$ is the product of its F1 scores across all knobs, *i.e.,* $\prod_r KnobValue\ ToScore(v_r)$, again following our independence assumption. Based on this, Algorithm 3 returns the cheapest k configurations whose accuracy is higher than a given threshold α'[2]. Note that the cost of a configuration c (line 10) may not be available from the preceding code since each knob is tuned independently, but it can be obtained from a one-time offline profiling because c's resource consumption is stable regardless of the video frame it is run on, as others have also observed [24, §6.2]).

[2]Since the accuracy estimates inevitably have some errors, we use a higher value of α' (by default, $\alpha' = 0.5 + 0.5 \cdot \alpha$ where α is the real threshold used in testing) to ensure the picked configuration yields an accuracy over α.

Comparing $c(v_r)$ against the golden configuration is costly, but it ensures that a good set of promising configurations is found. This is critical when profiling the full configuration space on a group leader at the start of a profiling window (line 5, Algorithm 2). Since this is done only once per profiling window per video group, the higher cost can be amortized. For all remaining video segments (line 7, Algorithm 2), the cost is too high as it applies per segment, so Chameleon sets all knobs other than r to lower default values in lines 5 and 6. The reason this works in practice is that the search space has already been reduced to the top-k most promising configurations, so finding a good relative ordering among them is sufficient, and is doable with lower default values. As long as the default values are not too low, the independence assumption approximately holds (extreme settings may violate the assumption, *e.g.,* a very small image size makes even relative comparisons difficult because no configuration detects any objects). Note that Algorithm 3 must still compute and threshold the final accuracy of each configuration.

Although line 4 loops over all values of a given knob, for some knobs (*e.g.,* frame rate, minimal area size), a lower value has no profiling cost because it can be extrapolated from higher values (*e.g.,* simply ignore frames to evaluate a lower frame rate). Also, since our knobs exhibit monotonically increasing/decreasing performance, we can stop the loop when performance is good enough (or bad enough, depending on the search direction). We used the former but not the latter optimization in our evaluation.

5.5 Practical considerations

We now cover two aspects that are critical in practice.

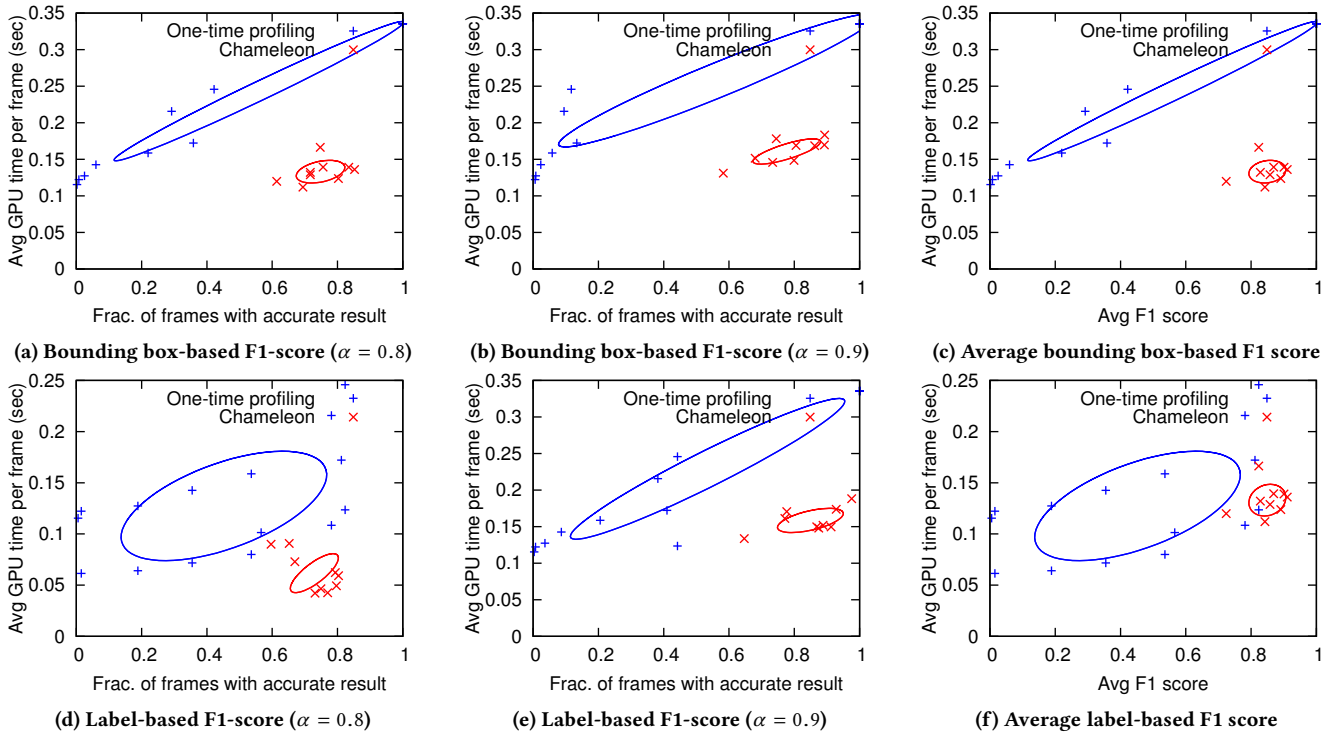
First, switching configurations in currently executing video pipelines is non-trivial, because the modules have to be designed to accept changes (*e.g.,* resolution). We build on prior work [32] where video modules constantly "listen" and prepare for any configuration updates. For example, when the configuration switches NN models, loading the new model into memory takes time (*e.g.,* up to 1 second). We must either factor this switching duration in our formulation or rely on pre-warming the memory with NN models.

Second, we believe it is best to use separate compute resources for profiling (separate from the compute used by the pipeline) to avoid disruptions to the live analytics. The recent trend in "serverless computing" [3, 4] makes it simple to run profiling tasks without explicitly provisioning VMs.

6 EVALUATION

We evaluate Chameleon on a dataset of video streams captured by multiple real-world traffic cameras over a duration of 24 hours. Our key findings are the following.

- Chameleon achieves significantly better inference accuracy and lower resource consumption than a baseline of profiling once upfront (§6.2).



(a) Bounding box-based F1-score ($\alpha = 0.8$)
(b) Bounding box-based F1-score ($\alpha = 0.9$)
(c) Average bounding box-based F1 score
(d) Label-based F1-score ($\alpha = 0.8$)
(e) Label-based F1-score ($\alpha = 0.9$)
(f) Average label-based F1 score

Figure 10: *Chameleon (red) consistently outperforms the baseline of one-time update (blue) across different metrics in Pipeline A. Each dot represents the results of running each solution on one hour of video from five concurrent video feeds. The graphs also include 1-σ ellipses [20] to mark the performance variance of each solution.*

- By leveraging the temporal persistence of configurations' accuracy, Chameleon reduces profiling cost by focusing on the top-k best configurations for most of the time (§6.3).
- By leveraging the spatial similarities across video cameras, Chameleon amortizes the cost of profiling across similar cameras with minimal reduction in accuracy (§6.4).
- Chameleon further cuts profiling cost by profiling independent configuration knobs separately (§6.5).

6.1 Dataset and setup

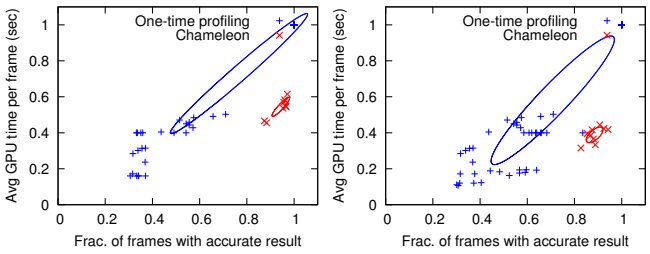
We use a dataset of video streams from five traffic video cameras deployed in different intersections in a metropolitan area (Bellevue, WA). The exact content of the video feeds varies significantly over time and across space. For instance, day time has more objects and intermittent traffic congestion, while at night object appearances are more spread out over time. Spatially, as well, two cameras in downtown areas show more cars at slower speeds than the other cameras deployed in suburbs (Figure 7 shows two screenshots). In addition, all cameras exhibit transient car motion patterns when traffic lights change. Despite their heterogeneity in content, we show that Chameleon can opportunistically leverage temporal persistence and spatial similarities between the cameras to achieve efficient online configuration adaptation. To obtain a representative dataset, we sampled 120 video clips across 24 hours from each camera, and used their original encoded MP4 format (1280×960p resolution, 30fps frame rate, and 150 seconds length) as the input to Chameleon.

We also used a different set of videos to generalize these results. The videos were taken from 10 cameras deployed in an indoor cafeteria area (with prominently posted notices) over a period of 3 days. The original MP4 videos are 1920×1080 in resolution and 25 fps in frame rate. Their content includes different patterns of human movement, *e.g.,* more people moving before/after meal times than the rest of the day. To obtain a representative dataset, we sampled 90 video clips, 9 clips from each camera, across the 3-day period.

The video frames are streamed in chronological order to the video analytics pipelines, whose configurations are controlled by Chameleon. We used the following control knobs (§2.1): for Pipeline A: we used 5 levels of frame rate, 5 levels of image size, and 6 pre-trained object detection models [7]; for Pipeline B: we used 5 levels of minimum region size with detected motion, and 4 pre-trained classifier models [8]. The inference models are implemented in Tensorflow and are pre-trained on standard image datasets [23], and the switching of video frame rate and resolution is done by FFmpeg [10].

6.2 End-to-end improvement

We start with the end-to-end improvement of Chameleon over the baseline of a one-time update (profiling configurations once at the beginning of a video stream). Figure 10



(a) Bounding box-based F1-score (b) Label-based F1-score

Figure 11: *Chameleon (red) outperforms the baseline of one-time update (blue) on Pipeline B (α=0.8).*

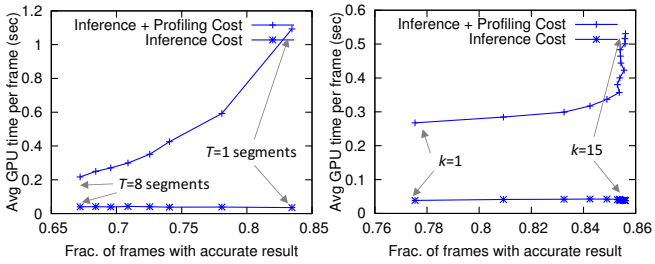
	Resource (avg. GPU time in seconds per frame)			Accuracy (frac. frames w/ bounding box F1 score > 0.8)		
	Chameleon	Baseline	Improv.	Chameleon	Baseline	Improv.
A	0.24	0.60	**60.2%**	0.65	0.65	1%
B	0.21	0.28	24.6%	0.62	0.28	**119%**
C	0.33	0.52	**37.4%**	0.72	0.70	3%
D	0.41	0.67	**38.6%**	0.71	0.69	2%

Table 1: *Chameleon improves accuracy and resource consumption on another dataset of indoor video cameras.*

shows that for Pipeline A, Chameleon consistently outperforms the baseline along resource consumption and several accuracy metrics for different values of the accuracy threshold α. Specifically, we use *bounding box-based* accuracy, where we compute accuracy based on specific locations of objects on the image, and also *label-based* accuracy, where we only compare the *set* of objects on a frame (and ignore locations). Note that the resource (*i.e.,* GPU) consumption includes both profiling of different configurations and running the best configuration to get inference results. The data points are visually summarized with 1-σ ellipses, which show their maximum-likelihood 2-D Gaussian distribution [20]. For most of the frames, Chameleon partitions the five cameras into two groups, one with three cameras in the suburb area, and one with two cameras in the downtown area.

We observe improvement on three fronts. (1) Chameleon achieves 20-50% higher accuracy with the same resource consumption, which suggests it could benefit resource-constrained settings (*e.g.,* edge or mobile devices). (2) Chameleon achieves 30-50% reduction in resource consumption (a 2-3x speed up) while achieving almost the same accuracy as the baseline, which could save capital costs when resources are elastic but expensive (*e.g.,* cloud VMs). (3) Finally, Chameleon not only improves performance on average, but also reduces performance variance: Chameleon's 1-σ ellipses[3] are remarkably smaller than those of the baseline. This is because Chameleon continuously adjusts its configuration over time, whereas the baseline is sensitive to the starting points of the video.

[3]In many cases, the baseline selected the expensive golden configuration (top right corner of the graphs), causing the ellipses to shift.



(a) Impact of profile window length (b) Impact of top k

Figure 12: *Impact of key parameters in temporal incremental update. The accuracy metric is the label-based F1 score with accuracy threshold α = 0.8.*

Notice that Chameleon still has non-trivial room for improvement compared to the optimal (idealized) performance, *i.e.,* periodic updating with zero profiling cost (Figure 4).

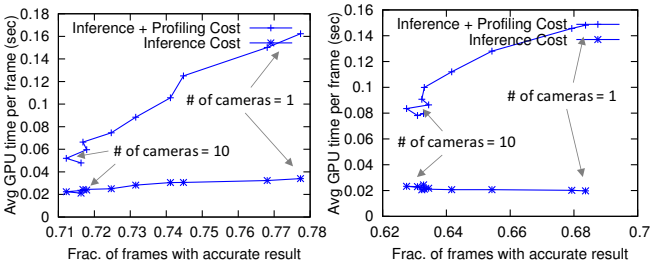
We observed similar results in Pipeline B. Figure 11 compares Chameleon and the baseline on two accuracy metrics in Pipeline B. Chameleon gives accurate results (F1 score over 0.8) on over 90% of the frames, while the baseline suffers from higher resource consumption or low accuracy, and substantial performance variance.

To generalize our end-to-end evaluation, Table 1 shows the improvement of Chameleon on another dataset of 10 camera feeds (see §6.1). Based on the cameras' geographical proximity, the cameras are partitioned into four groups (rows). The table compares Chameleon with a baseline that uses one-time profiling on each camera feed. We see that, for three groups (A, B, D), Chameleon reduces the GPU consumption by 37.4% to 60.2% without sacrificing accuracy, and in the other group (C), Chameleon increases accuracy by 119% while still managing to reduce GPU usage by 24.6%.

6.3 Impact of temporal incremental updates

Next, we microbenchmark the impact of individual components in Chameleon. We start with temporal incremental updates (§4.1), and investigate two of its key parameters: the profile window size and the size of the top-k set. First, we fix the parameters of Chameleon and only change the profile window size to see the impact of updating the top-k configurations less often. Figure 12a confirms our intuition (§4.1): when we start increasing the profiling window size from one segment (the top right corner), we see a fast drop in profiling cost (the gap between the two curves) relative to the degradation in accuracy, until some "knee point" (around 3-5 segments) after which further increases bring diminishing savings in cost while reducing accuracy.

Another key factor in temporal incremental updates is the size of the top-k configuration set. Intuitively, using a larger set introduces more overhead to check the set's accuracy in each segment, but tolerates more temporal variance because the best configuration is more likely to be found within the set. Figure 12b quantifies this tradeoff as we gradually increase k from 1 (bottom left) to 15 (top right). We observe a



(a) More similar cameras (b) More cameras, not all similar

Figure 13: *The benefit of having more cameras.*

good balance at around $k = 5$, below which top-k seems not able to tolerate transient temporal variance (*i.e.,* accuracy degrades), and above which the increased cost of checking the top-k set's accuracy does not yield much benefit.

6.4 Impact of cross-video inference

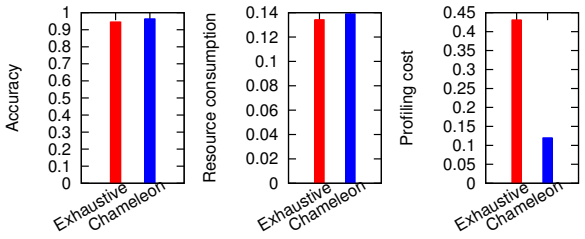
The second insight behind Chameleon is the existence of multiple similar cameras over which we can amortize the profiling cost. Figure 13 shows the benefits of having more cameras in two cases: when the cameras are similar, and when the cameras are not all similar. Naturally, in the former case we expect the cameras to share the same set of good configurations, so the cost of profiling should drop almost linearly with the number of cameras. Indeed, Figure 13a corroborates this intuition. We take ten video feeds[4] covering the same day-time hour, and incrementally add one camera to the test at a time. Chameleon automatically groups these cameras into one group (using the logic described in §5.3), and achieves a linear reduction in cost with only small reduction in accuracy. In contrast, during the hours when the cameras are less similar, we expect to see less reduction of profiling cost, since only a subset of cameras can share the profiling cost. This is exactly what happened in Figure 13b: Chameleon groups the cameras into two (sometimes three) groups, so even though there are 10 video feeds, the saving in profiling cost is lower than that in Figure 13a.

As we group more cameras, notice that the accuracy drops (albeit by less than 10%). This is because cameras in the same group might have different characteristics, so sharing the same configuration among them leads to lower accuracy than customizing the configuration for each one. The drop in accuracy is also partly due to our very simple camera-grouping algorithm (§5.3), which bounds the maximum discrepancy between the accuracy of a randomly chosen configuration on different video feeds in the same group. It is quite possible that a more sophisticated algorithm for grouping cameras would maintain higher inference accuracy.

6.5 Impact of reduced configuration space

The last key technique in Chameleon is to reduce the cost of a single profiling of the configuration space, by profiling each knob separately. The reduction in profiling cost is obvious, but the impact on accuracy and inference cost of the selected configuration is less evident. In Figure 14, we compare the configurations found by Algorithm 3 (based on the assumption that knobs are independent) to an exhaustive search, along three metrics: accuracy (Figure 14a), inference cost of the configurations picked by each algorithm (Figure 14b), and the profiling cost of executing the algorithms (Figure 14c). We can see that the configurations picked by Algorithm 3 are almost as good (in accuracy and inference cost) as the result of running an exhaustive search, while achieving an enormous reduction in profiling cost. Note that the "exhaustive" method is not optimal, because it only profiles the first second of each segment (not every second).

[4]We created 10 video feeds from 5 cameras by horizontally splitting the view of each camera into two non-overlapping video feeds.



(a) Accuracy (b) Inference cost (c) Profiling cost

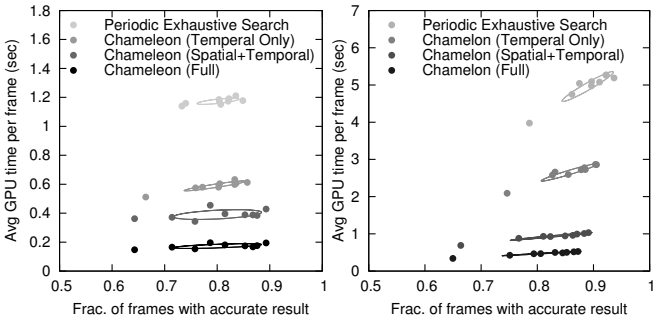
Figure 14: *Comparing Chameleon's profiling algorithm (Algorithm 3) with the result of an exhaustive search.*

6.6 Contribution of each component

Finally, we investigate the contribution of individual techniques in Chameleon, by incrementally adding one technique at a time (temporal incremental update, spatial cross-camera inference, and leveraging knob independence). Figure 15 shows the performance of the full Chameleon solution as well as some intermediate design points for the two pipelines we studied. In both pipelines, we see that each step brings significant reduction in cost at a relatively small drop in accuracy. Temporal incremental updates reduces profiling cost by about ~50%, cross-camera inference by an additional ~30-60%, and knob independence by another 40-60%.

7 RELATED WORK

Video processing optimization: Several previous papers have considered optimizing video processing pipelines by either adjusting the configuration knobs or training specialized NN models. VideoStorm [32] first profiles each video query running in a cluster and then adjusts its configuration to achieve the right balance between accuracy, processing delay, and resource demand. NoScope [24], MCDNN [16], and Focus [22] all process streaming or offline video using various NNs to detect objects, and recognize people and text. One of the core techniques in all three papers is training specialized NNs based on objects that typically appear in a specific video stream. For example, instead of a NN that can classify across 1000 objects, they train a much smaller (and more efficient) one for the top 20 objects. While each of



(a) Pipeline A **(b) Pipeline B**

Figure 15: *Contribution of individual components*

these papers reports significant improvements in accuracy and/or resource consumption, they all profile and optimize the video queries only once at the beginning of the video stream. They do not report how the optimal profiles change over time and do not handle changes in video stream content. An exception is [29] which retrains the NN model to detect the set of popular objects as it changes over time. Two core contributions of Chameleon are demonstrating that optimal configurations do change over time, and providing an efficient technique for continuously adapting the profiles.

Finding optimal configurations: Chameleon periodically searches an exponentially large configuration space to find the optimal NN configuration for a video query. This is done, at a minimum, for the leader of each spatially-related group of videos. Several recent systems have also faced an exponentially large configuration search space in their problem domains [12, 19, 31, 34]. Ernest [31] uses optimal experimental design [28] to optimize the VM configuration of a job, while Cherrypick [12] uses Bayesian optimization [27] to find an optimal cloud configuration for general applications. Hill *et. al* [19] use Thompson sampling [11] to optimize the layout of a multi-component web page; they use greedy hill climbing to select the next layout, similar to Chameleon, but Chameleon exploits more independence structure and monotonicity. All of these works bound the cost of their configuration search (*e.g.,* by adding it as a constraint in the optimization problem), but these are still one-time or daily costs paid for the modeling task at hand. Some bandit algorithms address non-stationary settings (*e.g.,* [26]), but these are too inefficient at present.

Chameleon differs from these systems in two major ways. First, the optimal configuration for a video is non-stationary, requiring frequent (every few seconds) re-profiling that must keep up with a real-time video feed. This puts tremendous pressure on keeping the profiling cost low. Second, Chameleon reuses optimal configurations across related video feeds. These differences lead to our greedy hill climbing approach, which avoids any computationally-expensive modeling.

8 DISCUSSION AND FUTURE WORK

Network bandwidth: Besides computational cost, which is the focus of this paper, network bandwidth is also an important resource in video analytics. With the increasing trend of running analytics across smart cameras [1, 5, 13] and the cloud, the network will start becoming a scarce resource in video analytics systems. The choice of configuration of a video analytics pipeline has implications on the network usage too; *e.g.,* one can save bandwidth by streaming the video at a low frame rate or resolution if it still maintains high inference accuracy. While Chameleon optimizes the computational cost of video analytics, its techniques addressing dynamic variations in the profile using spatial and temporal cross-camera correlations will likely carry over when considering the network. The problem, however, will be one of multiple resources and hence will require techniques for joint consideration of these resources.

Profiling on the edge: Chameleon's design relied on a separate cluster for its periodic profiling so as to avoid any disruptions to the live video analytics pipeline. As we move to the scenario of edge camera analytics [13], such a separation would be hard to achieve. The camera's compute resource, already limited, would be insufficient if it also has to accommodate the demand for periodic and quick profiling. At the same time, periodically recording and shipping video clips to the cloud for profiling may also overload the network. Arriving at the right design choice for edge camera analytics will be an important problem going forward.

Triggering the profiling: An unstudied aspect of resource-accuracy profiles in our paper is the periodicity with which their values change. While our solution relied on setting a pre-fixed time interval for profiling (and then saving its cost), we can save on profiling costs even further if we can predict the need to re-profile. Conceivably, we could use vision techniques like scene understanding to trigger fresh profiling. We believe this requires work at the intersection of systems, machine learning, and computer vision.

9 CONCLUSION

In this paper, we argue that video processing pipelines have to be adapted over time, otherwise they might achieve very low levels of accuracy. However, a naive re-profiling is prohibitively expensive. Instead, we present Chameleon, a system that uses several techniques to dramatically reduce profiling cost and also improves accuracy.

ACKNOWLEDGEMENTS

We appreciate the feedback by the anonymous SIGCOMM reviewers and our wonderful shepherd, Romit Roy Choudhury. Ion Stoica is supported by the NSF CISE Expeditions Award CCF-1730628, and in part by DHS Award HSHQDC-16-3-00083, and gifts from Alibaba, Amazon Web Services, Ant Financial, Arm, CapitalOne, Ericsson, Facebook, Google, Huawei, Intel, Microsoft, Scotiabank, Splunk and VMware.

REFERENCES

[1] Amazon aws deeplens. https://aws.amazon.com/deeplens/.

[2] Artificial Intelligence Surveillance Cameras Security. https://www.theverge.com/2018/1/23/16907238/artificial-intelligence-surveillance-cameras-security.

[3] AWS Lambda. https://aws.amazon.com/lambda/.

[4] Azure Functions. https://azure.microsoft.com/en-us/services/functions/.

[5] Google clips. https://store.google.com/us/product/google_clips?hl=en-US.

[6] New Search Engine Revolutionizes Video Surveillance. https://i-hls.com/archives/80734.

[7] Tensorflow detection model zoo. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md.

[8] Tensorflow-slim image classification model library. https://github.com/tensorflow/models/tree/master/research/slim.

[9] Yolo. https://pjreddie.com/darknet/yolo/.

[10] FFmpeg. http://ffmpeg.org/, 2000–2018.

[11] S. Agrawal and N. Goyal. Analysis of thompson sampling for the multi-armed bandit problem. In *25th Conference on Learning Theory (COLT '12)*, pages 39.1–39.26, 2012.

[12] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)*, pages 469–482, 2017.

[13] G. Ananthanarayanan, V. Bahl, P. Bodik, K. Chintalapudi, M. Philipose, and L. Ravindranath. Real-time video analytics - the killer app for edge computing. In *IEEE Computer*, 2017.

[14] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[15] R. Girshick. Fast r-cnn. *arXiv preprint arXiv:1504.08083*, 2015.

[16] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 123–136. ACM, 2016.

[17] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *IEEE International Conference on Computer Vision (ICCV), 2017*, pages 2980–2988. IEEE, 2017.

[18] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[19] D. N. Hill, H. Nassif, Y. Liu, A. Iyer, and S. Vishwanathan. An efficient bandit algorithm for realtime multivariate optimization. In *23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*, pages 1813–1821, 2017.

[20] W. E. Hoover and M. Rockville. *Algorithms for confidence circles and ellipses*. Citeseer, 1984.

[21] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[22] K. Hsieh, G. Ananthanarayanan, P. Bodik, P. Bahl, M. Philipose, P. B. Gibbons, and O. Mutlu. Focus: Querying large video datasets with low latency and low cost. *arXiv preprint arXiv:1801.03493*, 2018.

[23] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. *CoRR*, abs/1611.10012, 2016.

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

[25] F. Loewenherz, V. Bahl, and Y. Wang. Video analytics towards vision zero. In *ITE Journal*, 2017.

[26] H. Luo, A. Agarwal, and J. Langford. Efficient contextual bandits in non-stationary worlds. *CoRR*, abs/1708.01799, 2017.

[27] J. Mockus. *Bayesian approach to global optimization*. Kluwer, Dordrecht, 1989.

[28] F. Pukelsheim. *Optimal Design of Experiments*. John Wiley & Sons Inc., New York, 1993.

[29] H. Shen, S. Han, M. Philipose, and A. Krishnamurthy. Fast video classification via adaptive cascading of deep models. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017.

[30] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.

[31] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, and I. Stoica. Ernest: Efficient performance prediction for large-scale advanced analytics. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16)*, pages 363–378, 2016.

[32] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman. Live video analytics at scale with approximation and delay-tolerance. In *NSDI*, volume 9, page 1, 2017.

[33] T. Zhang, A. Chowdhery, P. V. Bahl, K. Jamieson, and S. Banerjee. The design and implementation of a wireless video surveillance system. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pages 426–438. ACM, 2015.

[34] Y. Zhu, J. Liu, M. Guo, Y. Bao, W. Ma, Z. Liu, K. Song, and Y. Yang. Bestconfig: tapping the performance potential of systems via automatic configuration tuning. In *Proceedings of the 2017 Symposium on Cloud Computing*, pages 338–350. ACM, 2017.