AdaPyramid: Adaptive Pyramid for Accelerating High-resolution Object Detection on Edge Devices

Xiaohang Shi, Sheng Zhang, *Senior Member, IEEE,* Jie Wu, *Fellow, IEEE,* Ning Chen, Ke Cheng, Yu Liang, and Sanglu Lu, *Member, IEEE*

Abstract—Deep convolutional neural network (NN)-based object detectors are not appropriate for straightforward inference on high-resolution videos at edge devices, as maintaining high accuracy often brings about prohibitively long latency. Although existing solutions have attempted to reduce on-device inference latency by selecting a cheaper configuration (e.g., choosing a more lightweight NN or scaling a frame to a smaller size before inference) or eliminating a background containing no object, they often ignore various high-resolution features and fail to optimize for those videos. We thus present AdaPyramid, a framework to reduce as much on-device inference latency as possible, especially for high-resolution videos, while achieving the accuracy demand approximately. We observe that the cheapest configuration to achieve the accuracy demand varies significantly across both different frames and different regions in a frame. The underlying reason is that object features (e.g., the location, size and category of objects) are more uneven in high-resolution videos, both temporally and spatially. Moreover, we observe that the object size presents a prominent hierarchical distribution in high-resolution frames. AdaPyramid thus partitions each frame hierarchically just like a pyramid and chooses a content-aware configuration for each region, which is adapted online based on the feedback. We evaluate the performance of AdaPyramid on a public dataset and our collected real-world videos. The obtained results show that under comparable accuracy to the state-of-the-art solutions, AdaPyramid can decrease inference latency by 40% on average, with up to 2.5× speed-up.

Index Terms—Edge computing, neural networks, video analytics, object detection, online adaptation

1 Introduction

Driven by fast-growing computational capability and data availability, deep convolutional neural network (NN)-based object detection technologies are developing rapidly, playing a pivotal role in the modern intelligent surveillance systems. Considering the privacy issue and the limited bandwidth to accommodate enormous data, videos ingested from the cameras are often analyzed on edge devices in many scenarios [1], [2], [3], [4], [5], [6]. However, edge devices are often restricted in the capacity of computation and storage and thus cannot match the massive resource demand of today's object detection tasks. On one hand, state-of-the-art NN-based models often have sophisticated structures, requiring massive matrix operations per inference. They are thus too computation-intensive for edge devices. On the other hand, high-resolution (e.g., 4K, 8K) cameras are much more pervasive nowadays and they can be bought at much lower prices [7], [8]. Performing inference on such high-resolution video frames incurs even more overhead with prohibitively high latency.

- *X.H. Shi, S. Zhang, N. Chen, K. Cheng, and S.L. Lu are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, 210023, China.*
  E-mail: {xiaohang, ningc, kecheng}@smail.nju.edu.cn, {sheng, sanglu}@nju.edu.cn.
- *J. Wu is with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA.*
  E-mail: jiewu@temple.edu.
- *Y. Liang is with the School of Computer and Electronic Information/School of Artificial Intelligence, Nanjing Normal University, China.*
  E-mail: liangyu@njnu.edu.cn.

To alleviate the high inference latency with edge devices, several acceleration solutions [9], [10], [11], [12], [13] have been proposed. Nonetheless, existing approaches are not sufficient in two aspects. Firstly, most of them cannot handle high-resolution videos well. This is because they often ignore the video features brought by high resolution and thus fail to take advantage of these features proactively to optimize their systems. Secondly, they rarely study how to reduce the on-device inference latency of online videos with an accuracy guarantee. The underlying reason is that the accuracy evaluation on the fly is non-trivial without the ground truth. Although it is widely adopted to choose the NN that is as accurate as possible to label the frames [14], [15], this method incurs too heavy an overhead to be practical, specifically on edge devices. Moreover, we also show in this paper (§4.3.1) that it is very hard to estimate inference accuracy precisely.

Distinctly from prior works, we present AdaPyramid, a framework to reduce as much on-device inference latency as possible, especially for high-resolution videos, while still achieving the accuracy demand approximately. To this end, we attempt to reduce the inference workload by choosing the cheapest configurations under the accuracy demand. Here, a configuration refers to a particular combination of knob (e.g., resolution, sampling rate, and NN model) values in a video analytics system and a configuration is cheaper if it incurs lower latency. We conduct extensive case studies and observe that the cheapest configuration to achieve the accuracy demand varies significantly across both different frames and different regions in a frame. We claim that it is a prominent character of high-resolution videos since high resolution often brings more uneven object features both temporally and spatially (§3). We thus adopt a ”partition+adaption” method. Specifically, we partition each frame into different regions and then choose a content-aware configuration for each region, which is adapted in an online manner.

Furthermore, we observe that the object size in frames presents a hierarchical distribution based on a visual law called perspective effect, i.e., the objects near the camera look larger while the ones far away from the camera look smaller. Since high-resolution videos can cover a larger geographic range, the size gap between the smallest object in the bottommost region and the largest one in the topmost region is greater. This observation brings opportunities for designing our efficient partition strategy.

In addition to the configuration choice, we also integrate the background elimination technique into AdaPyramid. Here, background means the regions containing no object, which are unnecessary to detect. In the COCO [16] validation set, the area of background occupies 57% of a frame on average [17]. This technique can thus cut down the inference workload significantly. To find the background, we need to obtain object locations. Additionally, the location, size and category of objects can also help profile the accuracy of NN models on a specific region, and this enables good configuration decisions. Therefore, it is beneficial to perceive the video content to obtain object features above. In our system, we obtain object features represented as the width, height and centroid coordinates of the bounding box based on prediction from previous frames. The key is to learn the motion of objects, which is the advantage of object tracking algorithms. For this reason, instead of devising a prediction method ourselves like previous works [9], [12], we propose to utilize existing tracking algorithms in the design of our object feature predictor.

We encountered several key challenges when designing the AdaPyramid system. Firstly, the frame partition comes before the configuration choice stage and affects the overall performance significantly. Despite its importance, the partition strategy is rarely studied except for the trivial method which partitions frames uniformly. Although Remix [13] proposes an exhaustive algorithm, its searching space is too huge to afford even when it is executed offline. AdaPyramid hence requires a partition guideline to reduce the searching overhead. However, how a specific partition plan influences the inference latency is very subtle. Secondly, it is non-trivial to choose the cheapest configuration under an accuracy demand in an online system. There exists an incompatible contradiction that the exact accuracy of a configuration can only be exposed after the actual inference while it’s impossible to evaluate all the configurations for each region due to prohibitive latency. Last but not least, we have to solve two key problems when designing the object feature predictor: 1) which object tracking algorithms [18], [19], [9], [20], [21] can be utilized and 2) how to integrate them into AdaPyramid. We thus make sufficient studies to address the aforementioned challenges.

Firstly, it is intuitive that the ideal frame partition plan is highly correlated with the dynamic video content. However, previous works [13] fail to characterize the content and utilize it proactively, thus they can only resort to exhaustive methods. The crucial problem is how inference latency is jointly decided by the video content and partition plan, after excluding the influence of configuration choice. Therefore, we make sufficient exploration addressing this, which is utilized to generate the partition guideline. To be more specific, when partitioning horizontally, the top portion of a frame should be partitioned into thinner regions than the bottom portion, just like the pyramid. Based on this guideline, AdaPyramid frees itself from exhaustive work and is able to give fast and effective partition plans.

Secondly, AdaPyramid answers how to approach the cheapest configuration under an accuracy demand in an online video analytics system. We conduct extensive measurement experiments and show the difficulty of profiling the detection accuracy of configurations. To fill the gap between the offline profile and the actual performance, AdaPyramid takes advantage of the system’s feedback to make continuous adaptations to the configuration choices. Moreover, it applies a binary search-based method to enable fast convergence to the ideal choice.

Lastly, we investigate the existing object tracking algorithms and find that the ones with the typical detection-based tracking structure are very suitable to AdaPyramid. We develop a general framework to integrate them into our system and present the rationale behind it.

Additionally, we are aware of multiple detailed technical problems. For instance, the unavoidable object omission problem incurs potential harm to the system, such as accumulated error. Additionally, AdaPyramid is based on object feature prediction and cannot handle new objects well. Various techniques are developed accordingly, which greatly improve the overall performance.

We have implemented AdaPyramid on commercial mobile platforms (e.g., Nvidia Jetson devices) in Linux as a plug-and-play extension to the object detection module of typical video analytics systems (e.g., intelligent surveillance systems).

To sum up, the main contributions of this paper are as follows:

- We propose AdaPyramid, a novel framework to reduce as much on-device inference latency as possible, especially for high-resolution videos, while achieving the accuracy demand approximately.
- We conduct extensive case studies and make various observations on high-resolution videos, which are utilized to design techniques in our system. Specifically, the ”partition+adaption” design is based on the prominent spatial and temporal dynamics while the pyramid-like partition strategy is inspired by the hierarchical object size distribution in frames.
- We propose a suite of techniques: 1) we develop a novel tracking-based solution to characterize video content, which enables the design of other techniques; 2) we reveal how inference latency is jointly influenced by the video content and partition plan and develop a pyramid-like partition strategy; 3) we answer how to approach the cheapest configuration online with an approximate accuracy guarantee through a binary search-based method; 4) we solve

some detailed technique problems, which greatly improve the overall performance.

- We implement our framework on the Nvidia Jetson platform. Evaluation results on PANDA 4K dataset show that AdaPyramid achieves up to 2.5× speed-up with comparable accuracy to the state-of-the-art object detection solutions.

2 Related Work

Since performing object detection on high-resolution videos is too computationally intensive for edge devices, many solutions have been proposed to accelerate the inference. Some of them are hardware-based. For example, various accelerators such as GPU, FPGA [22] and ASIC [23] are developed with specialized drivers, boosting the computing resources on edge devices. In this paper, we place our emphasis on software-based solutions, which can be broadly classified into three categories: configuration choice, background elimination and the "detect+track" framework.

Configuration choice. Model compression techniques [24], [25], [26], [27], [28], [29] generate lightweight NNs for inference speed-up. We can thus tune the knobs of videos (e.g., decreasing resolution or frame rate) or choose lightweight NNs to accelerate inference, which can be denoted as choosing cheaper analytical configurations. Nonetheless, such speed-up often comes with accuracy loss. This intrinsic latency-accuracy tradeoff is considered and balanced in many previous works [30], [31], [32], [10], [33], [34]. Nonetheless, these works are not customized for high-resolution videos and thus have much potential to improve. Remix [13] considers high resolution and adopts a nonuniform partition method similar to our work. However, its goal is to improve accuracy with a latency budget. Therefore, it is not suitable for those applications requiring accuracy guarantees. Moreover, Remix is devised based on the premise that NNs have a fixed input image size for inference, which cannot completely apply to the latest object detectors [35], [36], [37]. By contrast, our design is inspired by extensive observations of high-resolution videos. Additionally, Remix assumes that the video content of different regions is relatively static, and thus cannot handle videos with very dynamic content.

Background elimination. Background (i.e., regions containing no object) of frames requires no effort of inference. Therefore, the main process of this technique is to locate ROIs (regions of interest, i.e., the regions which are possible to contain objects) and then perform inference only on these regions. Various papers [38], [9], [3], [39], [12], [11], [40] propose different methods to locate ROIs. For instance, [9] reuses motion vectors of video codecs. [11] uses a traditional optical flow-based object tracking technique. [41] applies reinforcement learning, while [39], [42] develop DNN-based methods. Our work is based on object tracking in general. However, we do not intend to design a new method like existing works, but to propose a general framework to utilize existing multi-object tracking algorithms. In this way, the advantages of the latest algorithms can be integrated into our system.

"Detect+track" framework. We can decrease the average detection latency by replacing expensive object detectors with lightweight object trackers for some frames, instead of using detectors all the time. Many "detect+track" solutions [43], [44], [9], [45] are thus proposed where the detector is just applied every several frames, while the lightweight object tracking (e.g. lucus kanade methods) is used to give results alternatively in between. Although the detection latency can be significantly reduced, this framework is not suitable for the scenarios with high accuracy demand. It is very hard to maintain the accuracy since the prediction accuracy of lightweight object tracking methods is often limited, and the errors propagate and accumulate before the next inference of the object detector. Besides, this framework fails to handle the changes of object appearance, the occurrences of new objects, and the occlusions of objects well [11], which performs even worse if such phenomenons appear frequently. Moreover, this framework cannot be well applied to the scenarios with objects moving too fast as well [46]. We do not adopt this framework and performs NN inference on every frame to ensure the system performance when high accuracy is demanded.

Video content characterization. Video analytics applications often focus on the foreground of frames, such as pedestrians and vehicles. Video content characterization means obtaining the features (e.g., the location, size and category) of these objects for the future utilization in system optimization. For example, EdgeDuet [10] identifies whether a region potentially contains small objects by using the detection results of the last frame directly, and then offloads such regions with high compression levels. Remix [13] obtains the probability distribution of different object size levels in a region to predict detection accuracy, by performing the measurements from historical frames. ELF [12] develops a attention-based LSTM to predict where the objects in previous frames may appear in the current frame to estimate the workload of different regions for parallel offloading and inference. EAAR [9] reuses motion vectors of video codecs to predict the regions where objects are likely to appear.

Nonetheless, these works [10], [13], [12], [9] fail to exploit the potential of motion prediction in advanced object tracking algorithms [20], [21] for the online accurate characterization of the comprehensive fine-grained features, influencing the feature-based optimization. Moreover, they often ignore the object omission problem during the content characterization (see §4.2.3), thus hurting the detection accuracy when background elimination is incorporated. By contrast, AdaPyramid remedies the above insufficiency.

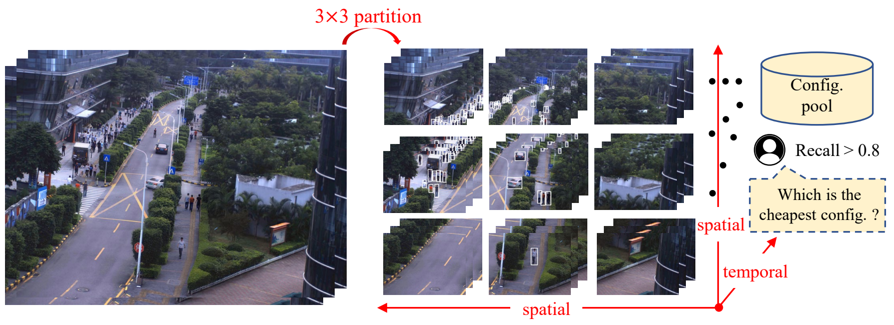
3 Motivations

In this section, We present the key observations of high-resolution videos to motivate us in the system design. To explore this, we conduct the following case studies.

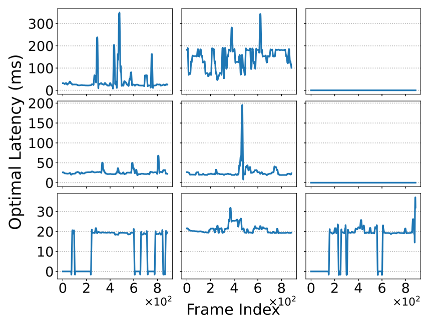
3.1 Case study setup

We use a traffic surveillance video with high resolution from PANDA 4K dataset[1], containing a road with moving pedestrians and vehicles. The resolution is 3,840×2,880 while the frame rate is 12. For a video, we need to choose the

1. It is generated by scaling the gigaPixel-level surveillance videos in PANDA dataset [47], [48] from 26,753×15,052 (32,609×24,457) to 3,840×2,160 (3,840×2,880).



(a) Case study setup.



(b) Case study result.

Fig. 1: Case studies for exploring features of high-resolution videos. In Fig. 1a, each frame is partitioned into 3×3 grids uniformly. We then perform object detection on these regions separately with the cheapest configuration (represented with black dots) to achieve the accuracy demand, using Jetson AGX Xavier. The latency of such cheapest configuration, denoted as the optimal latency, is plotted for each region of all the frames in Fig. 1b.

cheapest configuration with an accuracy demand anywhere and anytime if we want to achieve the lowest latency, which can be regarded as the optimal scheme. We thus make some explorations and observations from the spatial and temporal angles in this high-resolution video.

Fig. 1a shows the setup illustration. We partition each frame into 3x3 grids uniformly and perform inference on them respectively to detect pedestrians and vehicles. Detection results are marked with white bounding boxes. We use a typical edge device called Jetson AGX Xavier [49] as the inference device and Pytorch [50] as the inference engine. To obtain the ground truth of object detection, we use YOLOv5x as the oracle model to label the frames due to its nearly SOTA performance, instead of impractical manual labeling.

In this case study, the configuration is two-dimensional, which is a combination of NN and scaling ratio. The scaling ratio denotes the ratio of the width (or height) after scaling to the width (or height) before scaling. To construct the pool for configuration choice, we select YOLOv5n, YOLOv5s, YOLOv5m and YOLOv5l as optional NNs, which are four YOLOv5 [35] variants released officially. From variant v5n to variant v5l, the detecting capability gets stronger but with higher latency. We take the scaling ratio from 0.1 to 1 with a stride of 0.1, which corresponds to 10 different input resolutions. Finally, we get 40 configurations covering various detection capabilities in a fine-grained way. We then evaluate every configuration in the pool on every region of video frames. For each one of the total 9 regions, we pick the configurations with a Recall higher than 0.8 and choose the cheapest one among them. We denote the lowest latency of such cheapest configuration while meeting the accuracy demand as the optimal latency. For those regions containing no object, we regard the optimal latency as zero since they require no detection effort. We repeat this search for all the frames and plot the optimal latency for each region of all the frames in Fig. 1b.

3.2 Case study result.

In Fig. 1b, each sub-figure is plotted for one region in the corresponding place in Fig. 1a. We denote the region at the ith row and jth column as region (i, j) $(i, j \in \{1, 2, 3\})$. Since no pedestrian or vehicle appears in regions (1, 3) and (2, 3) all the time, latency is always zero, indicating that these regions are in the background and should be eliminated in the inference.

Observation 1: There exist prominent dynamics and unevenness in high-resolution videos. In the remaining regions except (1, 3) and (2, 3), we can observe prominent dynamics and unevenness. On one hand, latency varies greatly across different regions. Specifically, the optimal latency of top regions is higher than that of bottom regions by an order of magnitude. This indicates that the cheapest configuration to achieve the accuracy demand varies significantly across different regions in each frame.

On the one hand, latency fluctuates remarkably in each specific region. For instance, in the region (1, 1), the highest optimal latency during inference is more than 10 times the lowest. This indicates that the cheapest configuration to achieve the accuracy demand varies significantly across the same regions of different frames.

Therefore, if we adopt configurations as the unit of a whole frame, much more latency would be incurred to maintain the accuracy of the top regions. This motivates us to adopt the partition way and choose configurations for each one of the regions separately. Moreover, if we adopt a fixed configuration in all the frames, we would undertake more than 10 times the overall latency in the region (1, 1) to guarantee inference accuracy. This motivates us to adapt configurations over the frames to reach the optimum. Hence, we should adopt a "partition+adaption" design.

Observation 2: Object size distributes hierarchically in high-resolution frames. Object size decreases significantly from bottom to top, especially in high-resolution frames, presenting a hierarchical distribution. This trend brings opportunities to the design of a more sophisticated partition strategy. In this paper, such property inspires us to design the hierarchical pyramid-like partition strategy instead of the existing exhaustive search methods.

Discussion. To explain those observations above, we argue that they are prominent properties, especially in high-resolution videos. According to the perspective effect, the objects near the camera look larger while the ones far away from the camera look smaller. Since high-resolution videos often cover a larger geographic range, the size gap between the smallest and largest objects is greater. Therefore, the

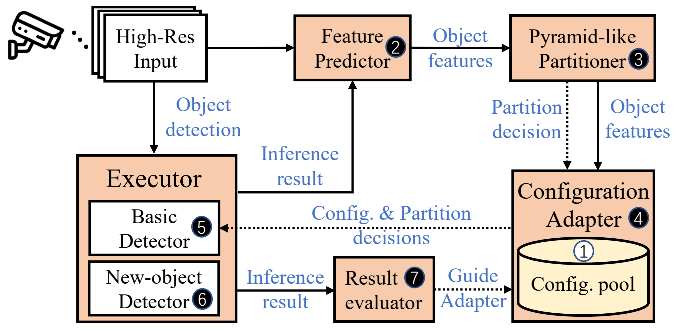


Fig. 2: AdaPyramid system overview.

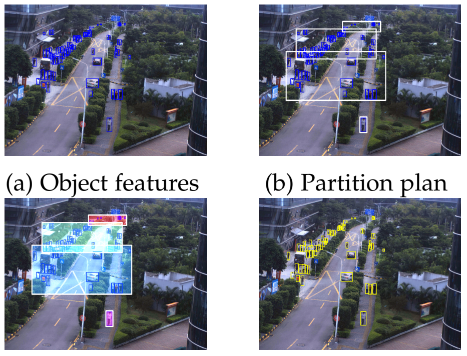


Fig. 3: An illustration of the processing pipeline for a frame in AdaPyramid. In Fig. 3a, object features marked with blue bounding boxes, are predicted for the current frame. Next, in Fig. 3b, the partition plan is generated by the pyramid-like partitioner. The partitioned regions are marked with white bounding boxes. After that, in Fig. 3c, the configuration adapter decides the configuration for each region, where their detection capabilities are marked with different colors. Finally, the per-region and new-object detection results are merged to produce the whole-frame inference result in Fig. 3d, which is marked with yellow bounding boxes.

object size decreases greatly from bottom to top, presenting strong unevenness and a hierarchical distribution. From the temporal aspect, the object size fluctuates in a larger range, resulting in remarkable dynamics in the time dimension. We also show the strong correlation between object size and the cheapest configuration to maintain accuracy later. That answers why the cheapest configuration to achieve the accuracy demand varies significantly both temporally and spatially.

4 System Design

4.1 Overview

AdaPyramid aims to reduce as much on-device inference latency as possible especially for high-resolution videos, while achieving the accuracy demand approximately. To realize this goal, it adopts a ”partition+adaption” design. Specifically, it first partitions the frame into several regions. It then eliminates the background and chooses the cheapest configuration to maintain accuracy for each region. Such workflow is repeated and adapted for each coming frame. Fig. 2 shows the system design of AdaPyramid while Fig. 3 is an illustration of the processing pipeline for a frame.

Feature Predictor ❷ (§4.2). AdaPyramid first captures a high-resolution frame from the camera. Feature predictor then predicts the features of objects in this frame, which is represented as the width, height and centroid coordinates. This predictor proposes a general framework for utilizing the motion prediction workflow of existing multi-object trackers. In the end, features are submitted to pyramid-like partitioner ❸.

Pyramid-like Partitioner ❸ (§4.4). This module decides how to partition a frame into several regions. It follows the guideline that the top portion of a frame should be partitioned into thinner regions than the bottom. It then eliminates the background of each region. The final shape is like a pyramid. Besides, the object features are further submitted to the configuration adapter ❹.

Configuration Adapter ❹ (§4.3). This module decides the cheapest configurations with an accuracy demand for each region across the frames. Starting from a conservative configuration obtained with the offline profile and object features, it continues to search based on the online feedback until the desired one is approached. Then the configuration choice and partition plan are submitted to the basic executor ❺ for inference.

New-object detector ❻ (§4.5). This module aims to detect new objects in the current frame, since the prediction-based detection above cannot handle objects never seen before. It is based on the observation that new objects just appear in specific regions of a surveillance video. Therefore, the inference is just performed on these regions, greatly reducing the overhead.

Result merging and feedback. Results from both per-region detections in the basic executor ❺ and new-object detection in the new-object detector ❻ are merged finally to produce the whole-frame result (Fig. 3d). This is then used to update trackers of feature predictor ❷ and guiding configuration adapter ❹.

4.2 Object Feature Prediction

4.2.1 Why AdaPyramid adopts object-level video content characterization?

Previous works [51] have shown that the performance of video analytics systems is sensitive to different features. Therefore, the characterization of video content may guide the design and optimization of these systems, which has shown up in the latest works [13], [12], [51], [10], serving as their foundation of performance boosting.

Instead of obtaining the probability distribution of different object size levels or the possible regions where objects may appear in some existing works [13], [9], [10], AdaPyramid aims to predict the features for every single object, called object-level video content characterization. This form contains fine-grained features of each object and is more informative intuitively. We adopt this manner since it contains extensive information, supporting the design of system techniques better. Moreover, it is predicted per frame to keep the freshness of features and can thus handle videos with dynamic content.

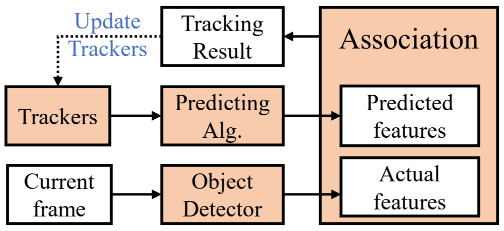


Fig. 4: The detector-based tracking structure of a typical multi-object tracking algorithm.

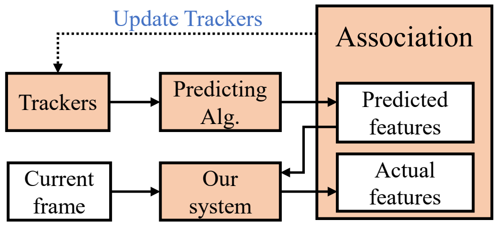


Fig. 5: A framework showing how to integrate a multi-object tracking algorithm into AdaPyramid.

4.2.2 *How to utilize the power of existing object tracking algorithms?*

To support configuration choice and background elimination techniques, AdaPyramid obtains features of objects in the current frame represented as the width, height and centroid coordinates of the bounding box. This is based on motion prediction of such objects in previous frames, which is well studied in existing object tracking algorithms [18], [19], [9], [20], [21]. Therefore, AdaPyramid utilizes their power to predict object features.

The goal of object tracking is to figure out whether the objects in the current frame are the same objects in the last frame, which requires associating the objects in these adjacent frames. We investigate the existing multi-object tracking algorithms sufficiently and find that those with a detection-based tracking structure (Fig. 4) fit AdaPyramid very well. Specifically, such algorithms first apply the object detector to perform inference on each frame, and then associate the detection results between adjacent frames to achieve the object tracking goal. To achieve the association, the tracker instances apply the specialized predicting algorithm (e.g., Kalman Filter) to obtain the predicted object features in the current frame. The prediction is based on the accumulated detection results of previous frames. The predicted object features are then associated with the actual detection results, obtained by actually performing inference on the current frame with the object detector, to generate the tracking result (e.g., object features with ID). In this process, the accumulated motion information in the tracker instances is also updated, given the association results.

Since the object detector module exists in both AdaPyramid and those tracking algorithms, it is possible to reuse the module for integration. Additionally, we need to decouple the predicting workflow (i.e., generating predicted features with trackers based on the predicting algorithm) out of the tracking structure to obtain object features in AdaPyramid. Fig. 5 proposes a general framework to integrate a tracking algorithm into AdaPyramid. Although different tracking algorithms have various concrete implementations and sometimes the modules are mixed up, they are separated in the logical aspect. We can thus easily perform the decoupling and integration. In Fig. 5, the predicted object features are input into AdaPyramid, which are utilized by our system to perform optimized object detection on the current frame for the actual object features. Finally, the actual features and predicted features are associated, and the association result is then utilized to update the trackers.

4.2.3 *How to handle the omitted objects hurting the system performance?*

Problem proposition. In the process of performing object detection on a video, objects are often omitted in the results. We present two reasons to account for this phenomenon. On one hand, the analytical configuration is too cheap to detect some objects that are very small or partially occluded. On the other hand, some objects are fully occluded by some obstacle when moving and are thus impossible to be detected. Since the feature prediction of objects is based on their locations in the previous frames, such omissions bring challenges to the correct prediction of the current frame. This problem is non-trivial to handle because it is hard to tell whether an object is omitted or just leaves the view field permanently.

Common practices [12], [9] often ignore this object omission problem, and make a prediction just for the objects which are successfully detected in the last frame. Specifically, if the predicted location of an object fails to be associated with that of a factually detected object, the corresponding tracker is discarded. This means that if an object is omitted in the last detection, it will also disappear in the feature prediction for the current frame. This method is intuitive but cannot be applied to AdaPyramid, which contains configuration choice and background elimination as two main components. Firstly, the predicted object features are incomplete for this simple discarding mechanism. Secondly, even if the omitted objects appear again in the following frames, they are identified as brand-new ones. Without the accumulated information in the trackers, their motion cannot be predicted very precisely. Since the configuration choice is guided by the predicted object features, it may thus be influenced due to the incomplete and inaccurate predicting result.

Even worse, the background elimination is highly dependent on the feature prediction since it aims to remove as large regions containing no object as possible. In this case, if an object omitted in the last detection fails to be predicted for the current frame, the region containing this object may be wrongly skipped during the inference. It is thus omitted again in the current detection. In this way, it may be missed in all the following frames. Fig. 6 provides an example. In Fig. 6a, target objects in the yellow bounding box can be detected initially and are then fully occluded by some trees in Fig. 6b. When they return to the view field (marked with a red bounding box) in Fig. 6c, they fail to be covered by the actual detection region (marked with a green bounding box) and are thus omitted. Here, the actual detection region refers to the regions which are left after the background elimination and are actually detected. Such irreversible, permanent omissions of objects in the detection may be accumulated as time goes on, incurring significant accuracy drop.

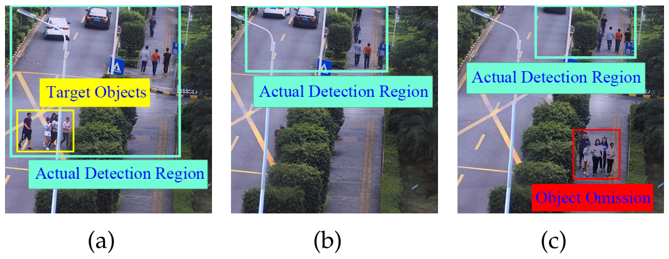


Fig. 6: Illustration for the potential problem with object occlusions. Figs. 6a-6c presents the change of the actual detection region before, during and after the occlusion occurs.

Problem solving. To alleviate this problem, we propose to reserve the trackers if the predicted object locations generated by them fail to be associated with the inference result. Such trackers continue to generate predictions in the following frames and are finally discarded only if the number of failing time in the associations exceed a threshold p. Through the reservation method, we provide a second chance for those trackers and expect them to succeed in the association when handling the following p frames. We can thus filter out most of the omitted objects and separate them from the ones leaving the view field permanently. As a consequence, the accuracy is improved significantly, particularly due to the sufficient mitigation of the accumulated object omission problem.

Although a high threshold p can help the omitted objects get more chances to be reassociated, this does not mean p should not be as high as possible. If the trackers corresponding to objects out of the view field are reserved for too many frames, the number of wrongly predicted features also increases. When eliminating the background, some regions which should have been eliminated are reserved since they contain such wrongly predicted features. Consequently, it hurts the inference latency. We thus conduct a tradeoff study in §5.3 to guide us to set p appropriately.

Moreover, the accumulated errors can be further eliminated by periodically applying the most expensive configuration for inference, providing the calibration signals in this close-loop system. However, additional inference overhead is also introduced. Since the tracker reserving design has suppressed the accumulated errors effectively, validated in Fig. 12a, we leave this calibration study as future work.

4.2.4 Cold start

Accurate prediction requires there being enough motion information accumulated in the tracker instances. However, the tracker instances are newly created when the system starts to capture frames, and thus cannot predict object features accurately for the detection workflow. To handle this cold start problem, in AdaPyramid, we apply the most expensive configuration on the whole frame for the first several seconds. In this way, the trackers are updated with the approximate ground truth for a sufficient number of times, and consequently can make accurate predictions then.

4.3 Configuration Adaptation

With an accuracy demand, this module aims to choose the cheapest configuration from a pool when performing inference on a specific region. Obviously, the foundation of making choices is to evaluate how a configuration performs on this region, which is called a profile. Although conducting the profile directly on the current frame is the most accurate method, its overhead is prohibitive since we have to make the choices online. Therefore, common practices [31], [30] often rely on offline profiles for the configurations. However, they rarely expose the gap between the offline profile and the actual performance. We then conduct extensive measurements on PANDA 4K dataset with Jetson AGX Xavier to evaluate the difficulty of obtaining a precise offline profile for both inference latency and accuracy. In the measurements, we use YOLOv5n, YOLOv5s, YOLOv5m and YOLOv5l as the candidate NNs, while YOLOv5x is used for the labeling. After that, we give our configuration adaptation algorithm inspired by the evaluation result.

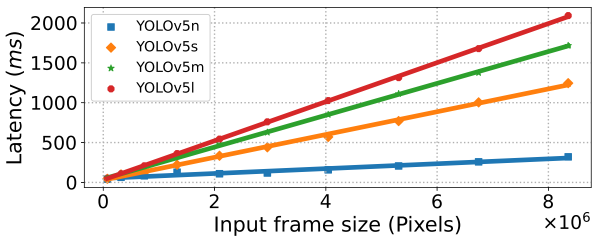


Fig. 7: latency of YOLOv5 series object detectors with different input frame sizes (number of pixels).

4.3.1 *Why is the offline profile not sufficient for configuration choice in an online video analytics system?*

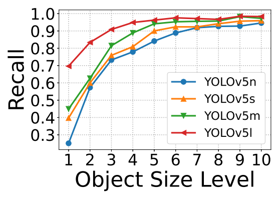
Measurement 1: Inference latency can be estimated precisely. We scale each frame to different sizes and measure the average inference latency on every frame size with candidate NNs. The scaling ratio ranges from 0.1 to 1 with a stride of 0.1.

We observe that the latency is almost consistent across different frames if they have the same size. This is rational since a typical NN performs the same operations on frames if the size of the input feature map is determined, regardless of the content. Furthermore, Fig. 7 shows that the latency of a NN increases linearly with the input size, denoting the number of pixels. This is because the time overhead of the main components in NNs such as convolution and pooling layers is linearly related to the size of the input feature map. Besides, some other components contribute relatively fixed overhead, such as some initialization operations and the classification network in Faster R-CNN. Therefore, the inference latency $L(\cdot)$ of a network n can be estimated precisely by fitting a linear expression with the measurement on a specific GPU, formulated as:

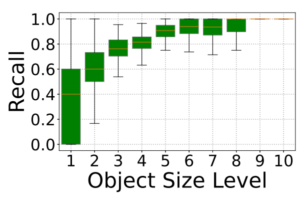
$$L(n, S) = k(n)S + b(n) \tag{1}$$

Here, S is input frame size while $k(n)$ and $b(n)$ are the coefficients related to network n.

Measurement 2: Inference accuracy cannot be estimated precisely. We have obtained object features from the predictor (§4.2) as the content characterization. Existing studies [12], [13], [51], [10] have proved that the accuracy profile is sensitive to object size and category. We thus measure the inference accuracy of candidate NNs on different sizes of pedestrians and show the results in Fig. 8.



(a)



(b)

Fig. 8: Accuracy of NNs when detecting pedestrians with different object size levels on PANDA 4K dataset. Fig. 8a shows the average accuracy of NNs while Fig. 8b further shows the YOLOv5s accuracy distribution on every object size level.

Object size is divided into multiple levels whose ranges are presented in Table 1.

Fig. 8a shows that accuracy decreases as object size gets smaller with a specific NN. Although we can profile the average accuracy with extensive measurements given a size level and a category, this doesn't mean accuracy can be precisely predicted. Fig. 8b shows that even with object size and category determined, accuracy still varies significantly across different frames, especially when object size is very small. For instance, accuracy ranges from 0 to 0.6 at size level 1, which is almost impossible to predict. This is because various factors, such as lightness, object occlusion and even whether a region is focused well, also affect the inference accuracy in addition to the factors we have considered already. Although introducing more factors is beneficial for precise prediction, it is still too complicated to construct a comprehensive model to characterize how accuracy is affected by them.

To summarize, it is not adequate to choose configuration just based on the offline profile due to the difficulty of accuracy estimation. We then present how to complement such insufficiency with the configuration adapter in AdaPyramid.

4.3.2 *Configuration pool* ①

Before giving the concrete design of configuration adapter, we first show how to construct the configuration pool. In AdaPyramid, the configuration is a combination of NN and scaling ratio. We denote the NN set as N and the scaling ratio set as R. The configuration pool P can be constructed as $N \times R$ ($\times$ is the Cartesian product). Although the configuration is two-dimensional here, we must mention that it can be easily extended to the multi-dimensional version.

Here, NN set N can be customized as the case may be, but it should better be able to cover every level of detecting capability. To construct R, we first denote the size (i.e., number of pixels) of the smallest object in the obtained features of a specific region as s. We then give the scaling ratio set R as $\{r_1, r_2, r_3, \cdots\}$ where $r_i = \sqrt{\frac{s_i}{s}}$ and $r_i \leq 1$. Here $s_i = (\frac{a+b}{2})^2$ pixels if object size at level i ranges from a^2 to b^2 pixels (e.g., $s_1 = 12^2$ pixels, $s_2 = 28^2$ pixels).

For a specific configuration $p = (n, r_i)$, we can offline estimate how it will perform on a specific region. We denote the size of this region as S. After the scaling operation, region size contracts to size ${r_i}^2 S$ while the smallest object size contracts to ${r_i}^2 s = s_i$, which is at the range of level

Size Level	1	2	3	4	5	6	7	$\cdots$
Min Size	0^2	24^2	32^2	40^2	48^2	56^2	64^2	$\cdots$
Max Size	24^2	32^2	40^2	48^2	56^2	64^2	72^2	$\cdots$

TABLE 1: The ranges of different object size levels in pixels, e.g., level 1 ranges from 0^2 to 24^2 pixels.

i. According to §4.3.1, inference latency can be precisely estimated as

$$L(n, {r_i}^2 S) = k(n){r_i}^2 S + b(n). \tag{2}$$

Since $k(n){r_i}^2$ indicates how much faster $L(\cdot)$ increases with S, it can be explained as the price of configuration p. To estimate the inference accuracy, we measure the average accuracy with Recall for each NN on different size levels of objects based on public datasets (COCO [16] dataset in our implementation) and denote the measured accuracy for level j and network n as $a(n, j)$. For example, according to this measurement, accuracy on the smallest object of the region above can be estimated as $a(n, i)$. We note that the estimated accuracy may not be very precise, but it can still be used for guidance.

4.3.3 *Configuration adapter* ❹

We now give the design of configuration adapter in AdaPyramid. We attempt to search for the cheapest configuration with an accuracy demand for a specific region, beginning from the one guaranteed to achieve this accuracy goal, denoted as the conservative configuration. We adopt this search direction from the expensive ones to the cheap ones out of particular consideration. We have learned from §4.2.3 that object omissions in one frame may introduce accumulated permanent omissions in the following frames. Therefore, we propose to keep the configurations applied in the search process to be more expensive than the desired result and this guides us to design our search direction.

Conservative start. We first propose how to choose the conservative configuration. Although the most expensive one in pool P can meet the accuracy demand, the process of searching is very time-consuming since the distance between the starting configuration and the result can be very long. To address this challenge, we can approximately regard the measured Recall of a network for a specific size level of objects as the probability of being detected. Therefore, although we have learned from §4.3.1 that inference accuracy cannot be estimated precisely, we can still guarantee that a specific size level of objects can be detected with high probability by setting a very strict accuracy demand. Since the smallest object is normally nearly the most difficult to detect, we choose the conservative configuration according to this object in order to guarantee the accuracy demand. Specifically, we choose the cheapest configuration from pool P whose estimated accuracy on the smallest object is higher than a strict threshold γ (0.95 in AdaPyramid) as the conservative configuration. Here, the smallest object size is provided by the predicted object features of the given region. We argue that such a setting of the conservative configuration can not only guarantee accuracy, but also reduce much search overhead by utilizing the predicted object features.

Starting from the conservative configuration, there still exists much potential for reducing latency by searching for cheaper configurations. For instance, in Fig. 8a, YOLOv5l achieves a Recall of 0.95 on level 5 objects but can still achieve 0.83 on level 2 objects. This indicates that even if we contract a region with level 5 objects to a quarter of its original size, we still have a high chance of detecting the objects in it, since an object at level 5 is approximately 4 times larger than an object at level 2. According to Eq. (1), scaling a region to a quarter of its original size means around 75% latency reduction. We then show the searching algorithm in AdaPyramid, which is a process of continuous adaptation across the frames.

Continuous adaptation. AdaPyramid searches the desired configuration by tuning the accuracy threshold γ. It then chooses the cheapest configuration whose offline estimated accuracy on the smallest object is higher than γ. Here, the threshold γ for the smallest object is just a knob for searching configurations with various capabilities. Additionally, we argue that although the estimated accuracy may not be very precise, it can reflect the relative relationship between the capabilities of different configurations. Therefore, a high γ can filter out relatively strong configurations, and vice versa. To realize fast searching, we develop a binary search-based method, which alternates between the proactive and passive stages.

AdaPyramid first enters the proactive stage. It decreases γ for every coming frame by a stride k set to k_1 initially. If no remarkable accuracy loss occurs, it continues with the same stride. Otherwise, γ is first recovered to the threshold of the last frame. After that, it resumes to search with half the initial stride, i.e., $\frac{k_1}{2}$. Each time the searched configuration shows remarkable accuracy loss, γ is first recovered and then resumes to be decreased with half the previous stride. When k decreases below a very low value (0.01 in AdaPyramid), we consider that the configuration is very close to our desired one and we switch to the passive stage.

In the passive stage, we first keep the configuration searched in the previous proactive stage in the next second, assuming video content shows no significant change in such a short interval. However, if remarkable accuracy loss occurs in this process, AdaPyramid will increase the accuracy demand continuously by a stride of k_2 until such a loss is eliminated. After that, it enters the proactive stage whose process is similar to the first one but with an initial stride of k_2. Here k_2 is smaller than k_1, as there is no need for drastic configuration adjustment after the first proactive stage. We set k_1 to 0.2 and k_2 to 0.05 in our implementation and the configuration can converge to the desired one within 12 frames in most cases, which is less than 1 second for videos with a frame rate of 12 (see §5.4).

It should be noted that if the searched configuration is not strong enough, object omission problem may be incurred along with the accumulated error. Fortunately, our predictor with the tracker reserving design (§4.2.3) can provide fault-tolerance for a relatively aggressive searching.

4.3.4 Result evaluator ❼

Since ground truth is unavailable when performing inference online, we cannot obtain the accuracy exactly. It is thus non-trivial to tell whether a searched configuration shows remarkable accuracy loss. In AdaPyramid, we approximately use the predicted object features as the ground truth since motion prediction for a short interval is accurate enough to serve as the evaluation labels. Besides, the fault-tolerant tracker reserving design (§4.2.3) further ensures the effectiveness of this approximation. Experiments (§5.3) also show that the accuracy evaluated with our approximate ground truth is close to the optimum. We consider a searched configuration as having shown remarkable accuracy loss if the evaluated accuracy is lower than a threshold α, namely the accuracy demand set by the users, including end users, service providers and application developers. Given the general importance of setting reasonable service quality guarantee (such as an accuracy demand, or a delay constraint), AdaPyramid allows users to achieve better experience by flexibly adjusting accuracy demands in specific applications, especially for providers and developers.

Therefore, although it is nearly impossible to obtain exact accuracy in practice, we can tune α to approximately satisfy our accuracy demand. Additionally, we should not set α to be very low, since it incurs significant object omissions in the current frame, which may introduce accumulated permanent omissions in the following frames (see §4.2.3).

4.4 Pyramid-like Partition

Frame partition comes before the configuration choice above, and determines the ceiling performance of configuration adaptation. Here, the ceiling performance means the lowest latency the system can achieve by choosing the cheapest configurations with an accuracy demand for each partitioned region. It is thus a very critical component.

Before presenting our partition design, a question must be answered: should partition plan be updated dynamically? Since the video content varies across frames, we should better update the partition plan for each coming frame. However, this strategy is not suitable for AdaPyramid. On one hand, making an update per frame incurs additional time overhead for this online system, hurting the overall performance. On the other hand, the configuration adaptation before is designed specifically for a fixed region. If the partition plan is updated very frequently, there isn't enough time for the configuration to converge to the desired version. Consequently, AdaPyramid adopts a fixed partition plan. We then provide our partition design.

4.4.1 Guideline of design

To guide the design, we propose that the key is to expose how inference latency is jointly decided by the video content and partition plan. However, the underlying relationship is very subtle and is rarely studied. Therefore, We conduct sufficient exploration of it here. In short, we utilize the hierarchical object size distribution in frames and develop an efficient partition strategy.

To control variables, we put the exploration under the fixed partition granularity. Here, partition granularity is a two-tuple denoted as (t_v, t_h), where t_v and t_h denote how many times the frame is partitioned vertically and horizontally respectively. For example, (0, 2) means partitioning a frame into regions with one column and three rows. Additionally, we assume that the cheapest configurations with an accuracy demand are chosen after the partitioning. We make this assumption to exclude the interruption of configuration choice on system performance.

We define resource waste for a region as the unnecessary latency generated by overly expensive configuration for larger objects, which is chosen to take care of the smaller ones. Specifically, we denote the cheapest configurations with the accuracy demand chosen for the smallest and largest objects as $p_1 = (n_1, r_1)$ and $p_2 = (n_2, r_2)$, and obtain resource waste R as $L(n_1, r_1) - L(n_2, r_2)$. We argue that by reducing as much resource waste as possible, the goal of reducing as much latency as possible is also achieved.

For further analysis, we denote the size of a given region as S. According to Eq. (1),

$$L(n_1, r_1) = k(n_1){r_1}^2 S + b(n_1). \tag{3}$$

We then denote the sizes of the smallest and largest objects in it as s_1 and s_2 respectively, and introduce $p_3 = (n_1, \sqrt{s_1/s_2}r_1)$, which means scaling the region with a ratio of $\sqrt{s_1/s_2}r_1$ and applies network n_1. From another point of view, p_3 can also be explained as scaling the region with a ratio of $\sqrt{s_1/s_2}$, which contracts the largest object from the size of s_2 to s_1, and then applies configuration $p_1 = (n_1, r_1)$. Since configuration p_1 on objects with size s_1 can achieve the accuracy demand, configuration p_3 on objects with size s_2 can also achieve the accuracy demand. Considering that p_2 is the cheapest configuration to achieve the accuracy demand on objects with size s_2, the latency of p_2 is not higher than that of p_3, i.e.,

$$\begin{aligned} L(n_2, r_2) &\leq L(n_1, \sqrt{s_1/s_2}r_1) \\ &= k(n_1)(\sqrt{s_1/s_2}r_1)^2 S + b(n_1) \\ &= k(n_1){r_1}^2 S \cdot \frac{s_1}{s_2} + b(n_1). \end{aligned} \tag{4}$$

Finally,

$$\begin{aligned} R &= L(n_1, r_1) - L(n_2, r_2) \\ &\geq L(n_1, r_1) - L(n_1, \sqrt{s_1/s_2}r_1) \\ &= (1 - \frac{s_1}{s_2})k(n_1){r_1}^2 S. \end{aligned} \tag{5}$$

Eq. (5) gives a lower bound of resource waste on a specific region. This implies that for the regions with the same width and height (same size S, of course), more resource waste is incurred in the top regions.

To explain this conclusion, we first present a visual law called perspective effect. It tells us that objects on top regions look larger, and vice versa. Moreover, for a particular category of objects, its visible height $h(\cdot)$ and width $w(\cdot)$ is linearly related to its centroid position on the vertical axis, denoted as y_{obj} [13]. Here, the direction of the vertical axis is from the top of the frame to the bottom. Therefore, the width of an object can be estimated as

$$w(y_{obj}) = p_w y_{obj} + q_w. \tag{6}$$

Here, p_w and q_w are the coefficients, which can be fitted with the frames collected from this specific camera. The height of the object can be estimated in the same way.

Now, on one hand, top regions contain smaller objects applied with expensive configurations, indicating that $k(n_1){r_1}^2$ is larger in top regions. On the other hand, in regions with the same height y_r, the width difference of the largest and smallest objects can be estimated as $p_w y_r$, which is irrelevant with objects. This means that the absolute width difference between the largest and smallest objects is the same across regions. That is to say, the relative width ratio between the largest and smallest objects is larger in top regions as the average object width is smaller there, and similarly for the relative height ratio, indicating that $\frac{s_1}{s_2}$ is smaller there. We can then obtain that resource waste R is larger in top regions. This inspires us to partition the top portion of a frame into thinner regions than the bottom, whose shape is like a pyramid.

4.4.2 Design of pyramid-like partitioner ❸

Based on the guideline that the top portion of a frame should be partitioned into thinner regions than the bottom, we present the design of our partition algorithm. Since the object size varies mainly in the vertical direction, AdaPyramid adopts the non-uniform strategy only when partitioning horizontally. When partitioning vertically, it partitions uniformly as there is no remarkable hierarchical object size variance in the horizontal direction.

Specifically, to partition horizontally, we let the relative width (or height) ratio between the smallest and largest objects be the same across different regions. We denote this ratio as β. Following the notations above and Eq. (6), for a region ranging from y_{up} to y_{down} vertically, the width of the smallest and largest objects can be estimated as $w(y_{up})$ and $w(y_{down})$. Therefore,

$$\beta = \frac{w(y_{up})}{w(y_{down})} = \frac{p_w y_{up} + q_w}{p_w y_{down} + q_w}. \tag{7}$$

Since we plan to partition horizontally for t_h times, we denote the t_h partition positions on the vertical axis from top to down as y_1, y_2, $\cdots$, y_{t_h}. According to the partition algorithm,

$$\frac{w(0)}{w(y_1)} = \frac{w(y_1)}{w(y_2)} = \cdots = \frac{w(y_{t_h-1})}{w(y_{t_h})} = \frac{w(y_{t_h})}{w(H)} = \beta. \tag{8}$$

Here, H denotes the height of the frame, while $w(0)$ and $w(H)$ are the estimated width of the smallest and largest objects in the frame. Thereby,

$$w(0) = \beta^{(t_h+1)} w(H). \tag{9}$$

Consequently, β can be computed as $\frac{w(0)}{w(H)}^{\frac{1}{t_h+1}}$. In turn, we can obtain the nth partition position on the vertical axis. Specifically, $w(0) = \beta^n w(y_n)$. Then, according to Eq. (6), y_n can be computed as $\frac{w(0)/\beta^n - q_w}{p_w}$.

We also compare our algorithm with the uniform strategy to evaluate its effectiveness. This evaluation is conducted on the traffic video in §3. The granularity is set to be (0, 2) to exclude the interruption of vertical partition, which is the same in both strategies. Additionally, the cheapest configurations to achieve the accuracy demand are chosen for each region of frames by searching exhaustively to exclude the interruption of configuration choice. Finally, the total detection latency for every frame is plotted in Fig. 9. We can see that our pyramid-like algorithm realizes lower latency on almost all the frames.

To explain this effectiveness, for the top regions applied with expensive configurations indicating larger $k(n_1){r_1}^2$, our algorithm achieves smaller $(1 - \frac{s_1}{s_2})S$ and thus prevents

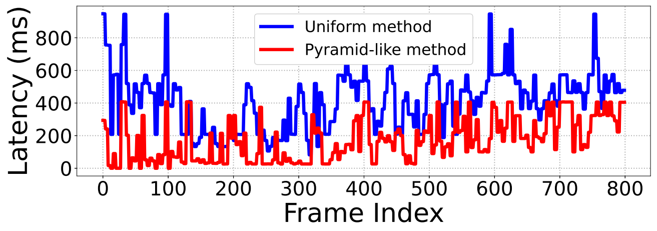


Fig. 9: Pyramid-like partition is better than uniform partition in terms of latency.

them from encumbering the overall latency (see Eq. (5)). Specifically, since we let the relative width (or height) ratio between the smallest and largest objects be the same across different regions, $\frac{s_1}{s_2}$ is the same as well. Moreover, the top regions are thinner, and thus have smaller size S. Consequently, this design follows the guideline above and gives the expected performance.

4.4.3 How to determine the partition granularity?

We have presented the partition algorithm with fixed partition granularity. So, how do we determine the granularity? Intuitively, a frame should be partitioned as finely as possible. In this case, the object size is very similar in each region, indicating little resource waste based on Eq. (5). However, a partition that is too finely grained may slice many objects into pieces, incurring significant accuracy loss. Therefore, an appropriate partition granularity should handle this trade-off carefully.

AdaPyramid determines granularity by evaluating different choices at the bootstrap stage. Specifically, we obtain the converged inference latency for each choice. Remember that configuration adapter enables fast convergence, we thus leave very little time (1 second in AdaPramid) for the evaluation of each choice. We observe in experiments that when t_v or t_h exceeds 2, AdaPyramid suffers significant accuracy loss in most videos. Such loss in turn forces more expensive configuration to meet the accuracy demand, thus leading to much higher latency. Based on this observation, we just evaluate a limited amount of 9 choices whose t_v and t_h range from 0 to 2. Therefore, the time cost for determining partition granularity is limited.

4.4.4 Background elimination

After determining the partition plan, AdaPyramid eliminates the background for each region. Remember that we have obtained the predicted features of objects, represented as the width, height and centroid coordinates of the bounding box. According to the features, we first assign each of them to the region whose overlap with the bounding box is the largest. We then adjust the boundaries of each region adaptively to exactly cover all the objects assigned to it. Additionally, some padding is reserved to handle the prediction deviation [12]. Fig. 10 shows an illustration. Note that this adjustment may introduce overlap between neighboring regions. Thereby, the objects in overlapping areas may be detected repeatedly. To handle this problem, we use the non-maximum suppression (NMS) technique to remove duplicated bounding boxes when combining the detection results of regions.

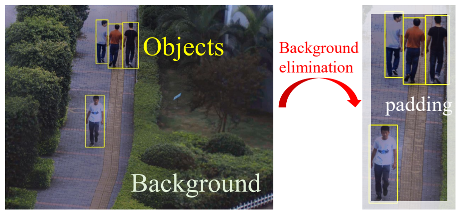


Fig. 10: An example of background elimination. The left sub-figure illustrates the original partitioned region while the right sub-figure is the result of background elimination. Padding is marked as a white band, which is added to make it tolerant for prediction deviation.

4.4.5 Discussion

AdaPyramid performs the pyramid-like partition by utilizing the offline profiled linear relationship (see Eq. (6)) based on perspective effect. We should note that this profiled relationship may not exactly apply to every object in the frame, especially in complicated scenes. For instance, in a shopping mall scene with multiple floors, the relationship profiled for objects on the same floor may not apply to the objects on different floors. However, this profile can still provide a good statistical result and experiments (see Fig. 16) show that the performance gain based on it is considerable. Besides, a more sophisticated partition strategy requires further understanding of the semantics in the frame, which may inevitably generate more system overhead and impractical for an online object detection system like AdaPyramid.

4.5 New-object Detection

In previous sections, the designing of the partition and configuration choice techniques is based on predicted object features. However, this prediction cannot handle new objects never seen before. Existing solutions often seek to locate them in the whole frame. For example, some works [12], [10] conduct NN-based detection for new objects. Although they have proposed to compress the frame (e.g., scaling to a lower resolution, or applying a lower compression quality) to reduce detection overhead, such reduction is often limited if there needs to be a guarantee that new objects should be located relatively accurately, especially when handling objects of very small sizes. In this case, the time cost is not efficient to improve the accuracy brought by new-object detection.

New-object detector❻. Fortunately, surveillance cameras are usually stationary in a fixed angle and position. We thus observe that new objects can just appear in some specific regions of a frame, bringing the chance of greatly reducing computation overhead by removing the regions in which it is impossible for new objects to appear.

Since the regions in which it is impossible for new objects to occur are camera-specific, we do the marking for each camera in the offline stage. It is thus a one-time effort. Since cameras are deployed at different positions with different angles, the camera-specific marked regions also vary significantly. However, we argue that the ratio of such regions to the whole frame is minor in most cases. This is because the regions are located with a great probability at some special positions, such as the frame boundaries and exits of buildings. Thereby, the detection work is limited.

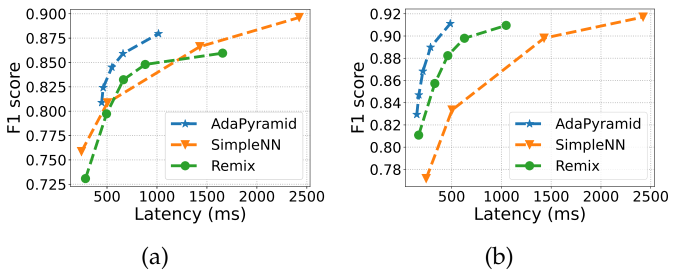


Fig. 11: Inference latency and F1 score on PANDA 4K dataset (11a) and collected videos (11b) with Jetson AGX Xavier.

With the offline marking, AdaPyramid then performs inference on marked regions. Since object omissions in the current frame may incur accumulated omissions later, we do not want to miss any new object. To this end, the new object detector works for each frame. Besides, we choose the cheapest configuration from pool P whose offline estimated accuracy is higher than 0.9. We set this high threshold to make sure the detection of new objects is accurate enough.

It should be noted that besides the NN-based methods, some traditional methods such as frame difference, optical flow and edge detection methods [11], [38] are also used to identify the regions containing objects. However, we propose that our method is orthogonal with theirs, since their methods can be applied to our marked regions for overhead reduction, instead of operating on the whole frame. Therefore, the performance gain can be combined.

5 EVALUATION

In this section, we evaluate the performance of AdaPyramid under different real scenes. We first compare the achieved latency of AdaPyramid with various baselines under different accuracy. We then conduct ablation studies to evaluate the performance gain from individual components.

5.1 Experiment Setup

5.1.1 Implementation

We implement AdaPyramid on a Jetson AGX Xavier with an 8-core Nvidia Carmel CPU and a 512-core Nvidia Volta GPU. The power mode is set to MAXN. We use Python for easy integration of deep learning applications. PyTorch is used as the inference engine on this mobile GPU for object detection. Additionally, we use the widely-used F1 score as the accuracy metric.

When constructing the configuration pool, we use {YOLOv5n, YOLOv5s, YOLOv5m, YOLOv5l} as the NN set. Before running our system, we load all the models into GPU memory. Therefore, there is no model switching overhead when performing inference with different models. Our candidate NNs consume around 2GB of GPU memory in total which is acceptable for Jetson AGX Xavier. For those edge devices with less GPU memory, we can customize the NN set for practical use. We then use YOLOv5x as the oracle model to do the labeling work, since its performance is nearly SOTA among all the known YOLO implementations.

To build our feature predictor, we use SORT [20] as the existing multi-object tracking algorithm with the typical detection-based tracking structure. This is chosen for its high tracking efficiency which consumes several milliseconds for one frame on CPU. We integrate it into AdaPyramid following §4.2.2 with some modifications in its source code. Although SORT is used in our system, we also allow users to apply other efficient MOT algorithms, whose integration method is similar.

5.1.2 Dataset

We evaluate AdaPyramid on the videos from PANDA 4K dataset for person and vehicle detection. PANDA is a public video dataset for large-scale, long-term, and multi-object visual analysis, captured by a gigapixel camera. It is composed by diverse real-world scenes (totaling 21 currently), including street, crossroad, basketball court, etc., captured from various camera angles and locations. Besides, it contains 75,600 frames with more than 200 objects in each frame on average. In this dataset, frame rate of videos are different. We thus perform downsampling on them and convert them to 12 fps videos universally.

5.1.3 Baselines

Our baselines include the following object detection solutions:

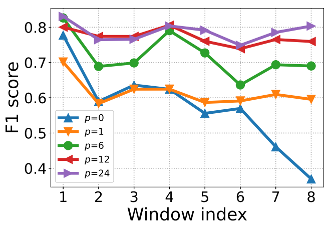
- SimpleNN. This is a widely adopted method, which performs NN-based inference straightforwardly on whole frames of videos. We respectively test YOLOv5n, YOLOv5s, YOLOv5m and YOLOv5l.
- Remix [13]. This is a state-of-the-art solution to improving inference accuracy with a latency budget. It assumes each NN has a fixed input frame size, while heavy models often have larger sizes than the light ones. However, YOLOv5 only has one training frame size, i.e., 640×640. Following the setting of Remix paper, we adjust the input size to 768×768, 896×896, $1,024 \times 1,024$ and $1,024 \times 1,024$ for YOLOv5n, YOLOv5s, YOLOv5m and YOLOv5l respectively. We then tune the latency budget and obtain different latency-accuracy tuples.

5.2 Overall Performance

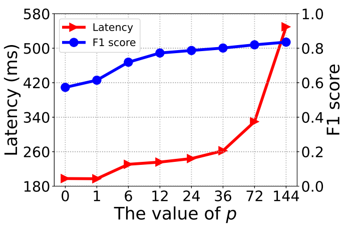
We present the overall performance of AdaPyramid and baselines in Fig. 11. For AdaPyramid, we tune α as 0.85, 0.87, 0.90, 0.92 and 0.95 respectively to achieve different latency-accuracy tradeoffs. Then, each point on the curves denotes achieved average accuracy and latency for frames from all scenes with one of the α settings above.

In Fig. 11a, AdaPyramid achieves the best performance compared to the baselines. Under similar accuracy, AdaPyramid can decrease latency by 40% on average with up to 2.5× speed-up. The performance gain is very prominent, particularly when we demand high accuracy. In this case, since more expensive configurations are used, there also exists more potential for reducing inference overhead. Compared to SimpleNN without the configuration adaptation and the background elimination, AdaPyramid reduces as much inference latency as possible by optimizing both aspects. Compared to Remix, the performance gain mainly comes from three aspects.

Firstly, AdaPyramid adapts the configuration based on the latest video content, i.e., object features updated per



(a)



(b)

Fig. 12: Evaluation of object feature predictor. Fig. 12a shows the average F1 score on successive time windows under different p values. Each time window contains 100 frames. Fig. 12b illustrates the accuracy-latency tradeoff as p changes.

frame. By contrast, Remix selects plans based on the performance estimation of them, which is calculated with the analysis on historical frames and may be out-of-date for high-quality plan selection.

Secondly, AdaPyramid eliminates the background with no object more thoroughly than Remix. Although AIMD-based selective execution is applied to skip the regions unlikely to contain objects, this approximate method cannot skip every background region and may wrongly skip the regions containing objects actually. Besides, there still exists much background in the regions containing objects, which Remix fails to handle.

Thirdly, Remix suffers from the densely partitioned regions, whose boundaries slice objects into pieces, hurting the accuracy remarkably. By contrast, AdaPyramid restricts very dense partition plans to alleviate the problem. Specifically, Remix sets the partitioned region sizes as the training input sizes of NNs. However, most NNs for object detection possess relatively small input sizes (e.g., 800×800 for RetinaNet, 1024×1024 for Faster R-CNN), leading to dense partition plans. For instance, in our settings, the partition granularity is more than 3×3 for 4K videos. Once the objects distribute relatively densely on the frames, too many of them will be sliced into pieces by the region boundaries. Even though Remix attempts to alleviate this by adding margins to the detection regions, the problem is not entirely solved in practice. From Fig. 11, we can observe that Remix fails to outperform SimpleNN when latency is either very low or very high. In such cases, Remix and SimpleNN both adopt very cheap or expensive configurations. However, Remix suffers significant accuracy loss in addition.

In addition to the public PANDA 4K dataset, we further evaluate the scalability of AdaPyramid on more videos and devices. Specifically, we deploy AdaPyramid in real world using Jetson AGX Xavier, and collect 4K videos for evaluation from a university campus at different times. They contain around 18,000 frames with about 15 objects in each frame on average. Compared with PANDA, the objects are thus relatively sparser in our collected videos. We present the performance of AdaPyramid and baselines in Fig. 11b. Due to the sparser distribution of objects in the frames, Remix can always outperform SimpleNN since the object slicing problem discussed above is greatly mitigated. Meanwhile, AdaPyramid can decrease latency by 61% on average with up to 3× speed-up, and still achieve the best performance, presenting its extensibility in various scenarios. We also evaluate AdaPyramid on a different Jetson TX2

TABLE 2: F1 score of the object feature predictor.

α	0.95	0.92	0.90	0.87	0.85
F1 score	0.85	0.85	0.86	0.85	0.85

TABLE 3: F1 score error of the approximate accuracy with the result evaluator.

α	0.95	0.92	0.90	0.87	0.85
F1 score error	0.04	0.02	-0.01	-0.02	-0.02

device. The results are illustrated in Fig. 13, showing 54% latency reduction on average with up to 3.1× speed-up on the datasets we use. We also observe that the trends of F1 score-latency curves are similar across different devices.

Next, we break down AdaPyramid and evaluate each key component, i.e., object feature predictor, configuration adapter, pyramid-like partitioner and new-object detector. Limited by space, we choose three typical scenarios when the results must be presented based on specific scenes.

5.3 Evaluation of Object Feature Predictor

To show the effectiveness of feature predictor, we first evaluate whether the object omission problem is solved. Recall that we have proposed in §4.2.3 that omissions in the current frame incur accumulated permanent omissions in the following frames. We solve this problem by reserving the trackers p times when they fail in the association on the first time. Now, we set p to different values, and plot the average F1 score on successive time windows in Fig. 12a. Here, a time window contains 100 frames.

Fig. 12a shows that such a proposed reservation method alleviates the problem. When our method is not applied, which means p is zero, accuracy drops significantly as time goes on. When p is set to 1, the catastrophic accuracy loss is eliminated. As p increases, accuracy also improves, since more omitted objects are well handled and are distinguished from those leaving the view field permanently. They thus incur no permanent omissions later.

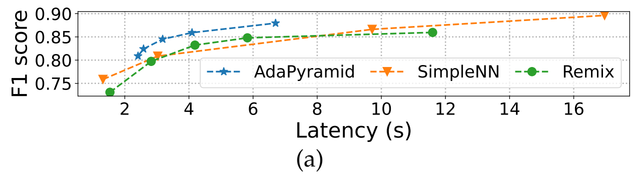
However, higher p hurts the inference latency. Fig. 12b presents the average accuracy and latency across different values for p. As p increases from 0 to 12, the F1 score goes up from 0.57 to 0.79. We notice that accuracy gain is limited when p is larger. This is because most omitted objects can be handled with a low value of p. Considering this tradeoff, we set p to 24 as it can mitigate the problem considerably but with a minor additional time overhead.

Next, we evaluate the prediction accuracy of feature predictor. Table 2 shows that with different accuracy demand α, the predicted object features can always maintain accurate enough to guide the configuration adapter and partitioner.

Furthermore, since the predicted object features are used as the approximate ground truth to online evaluate the detection results, we compare such approximate accuracy with the actual accuracy obtained by using the actual ground truth. Table 3 presents the F1 score error with different accuracy demand α, which is computed by subtracting the approximate accuracy from the actual accuracy. The results show that our approximation can approach the optimum and validate the effectiveness of the result evaluator.

5.4 Evaluation of Configuration Adapter

This module chooses the cheapest configurations with an accuracy demand, when performing object detection on a



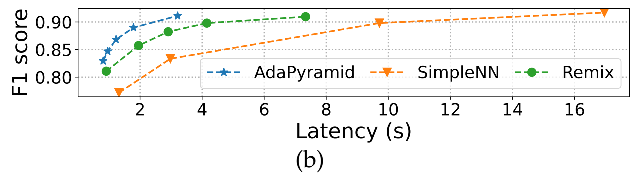


Fig. 13: Inference latency and F1 score on PANDA 4K dataset (13a) and collected videos (13b) with Jetson TX2.

video. This is implemented by adapting the choice both in a frame and across frames. Specifically, when adapting for a region, it starts from a conservative configuration and then searches for cheaper ones until the desired one is found. Fig. 14 illustrates the converging process in three different scenes (i.e., park, plaza and campus).

In Fig. 14, latency of the searched configuration reduces significantly in the first 20 frames. Starting from the conservative configuration decided initially, it achieves a 45-75% latency reduction in the following adaptation process. We can observe that convergence is achieved very fast, within 12 frames, i.e., less than 1 second for 12 fps videos, owing to our binary search-based adaption algorithm.

In the configuration adapter, α reflects the desired accuracy demand of the user. Fig. 15 shows latency-accuracy tradeoff by tuning the user demand, which is approximately guaranteed. As α increases, accuracy approaches the optimum achieved by the most expensive configuration, but also with a significantly increasing latency. Since a low α may introduce accumulated permanent object omissions (see §4.3.4), α is greater than 0.75 in AdaPyramid.

5.5 Evaluation of Pyramid-like Partitioner

Our pyramid-like partition algorithm is designed following the guideline that the top portion of frames should be partitioned into thinner regions than the bottom. We compare this algorithm with the uniform method in Fig. 16. The result shows that our design can achieve a 15% latency reduction on average with no accuracy compromise.

Additionally, the partition granularity affects performance significantly. Table 4 illustrates average latency with different granularity choices under the scene of the campus, where the lowest latency is achieved when granularity is (1, 1). In the scenes of the park and plaza, the lowest latency is achieved when granularity is (0, 2) and (1, 1) respectively. We observe that the best granularity for the park scene is finer than the rest when partitioning horizontally. This is because the resolution of the park video is 3,840×2,880 while the rest is 3,840×2,160. The hierarchical object distribution is thus more prominent in the park video. Moreover, the objects in it are relatively sparser than the rest with less probability of being sliced by region boundaries. It can thus tolerate denser partitions. From the experiments, we obtain that the best granularity in most videos is coarser than (2, 2). When granularity becomes finer, the object-slicing problem is more notable, thus hurting accuracy significantly. This

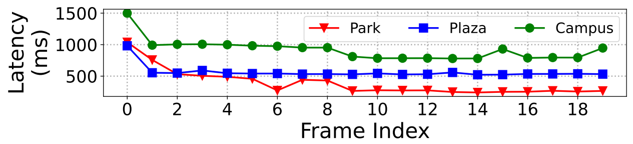


Fig. 14: Latency of configuration in the first 20 frames on three typical videos, i.e., park, plaza and campus, presenting fast latency convergence.

explains why we search for the best configuration from (0, 0) to (2, 2).

5.6 Evaluation of Fast New-object Detector

When handling new objects which first appear in a frame, new-object detector only performs inference on offline-marked specific regions in which new objects can occur. Compared to existing solutions that handle the total frame, this greatly reduces the time overhead since marked regions are limited in most cases. Fig. 17 shows the latency comparison between these two methods in three scenes. We use the same configuration choice method for both of them. Specifically, we choose the cheapest ones whose offline estimated Recall is higher than 0.9.

Results show that latency of the whole-frame method is too much for practical use unless accuracy is compromised. This is because the golden configuration has to be used on all three scenes to achieve the high accuracy demand. Our new-object detector is 10× faster since we only have2to handle marked regions whose sizes are much smaller. Moreover, configurations can be chosen based on the features of each region.

5.7 System Overhead

System overhead is minor in AdaPyramid, which mainly comes from three components: object feature predictor (with tracker update), configuration adapter, and result merging. Since our pyramid-like partition is a fixed strategy only requiring one-time effort, it is not covered in the analysis. Table 5 shows the average latency of these components as well as which compute unit they are running on (only the NN inference is performed on GPU). Since in most cases the total end-to-end latency is larger than 160ms, the system overhead is often less than 10% of it. It is sufficiently low to deliver main functions such as pyramid-like partition and configuration adapter for the significant latency reduction.

We observe that the object feature predictor and result merging consume more than 90% of the total system overhead. This is because videos in the PANDA 4K dataset often contain 100+ objects in each frame, while the overhead of both the motion prediction algorithm in the feature predictor and the NMS operation in the result merging is highly related with the number of objects. Therefore, when working on videos with fewer objects, the system overhead of AdaPyramid can be reduced accordingly.

5.8 Memory Footprint

We measure the memory footprint of AdaPyramid, which consumes at most 7.8 GB in the running process. We observe that the weights and immediate tensors occupy the majority portion (around 5 GB), since AdaPyramid loads all the NN models into GPU memory before running the system, which

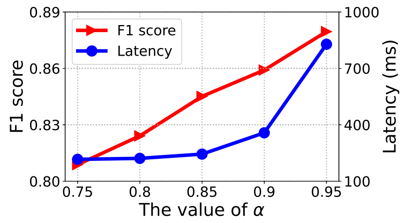


Fig. 15: Accuracy-latency trade-off as α value changes.

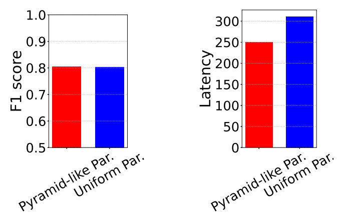


Fig. 16: F1 score and latency with uniform partition method and our pyramid-like partition method.

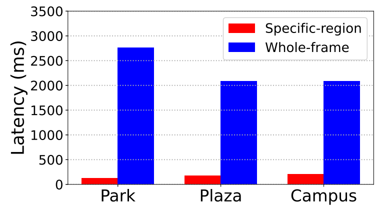


Fig. 17: Latency with the common whole-frame method and our specific-region method.

t_h / t_v	0	1	2	3
0	798	792	1291	1054
1	1085	636	960	948
2	1110	823	935	992
3	1310	879	1029	1179

TABLE 4: Latency with different partition granularity choices on campus video.

Components	Latency (ms)	Compute Unit
Config. adapter	1.29	CPU
Predictor	11.13	CPU
Result merging	3.49	CPU
Total	15.91	CPU

TABLE 5: Average latency of the system components and the compute units that they are running on.

will be utilized to perform object detection mixedly on the video frames. We adopt this straightforward method so far to get rid of the model switching overhead. However, a more sophisticated loading and unloading strategy for the NN models can be potentially more memory-efficient. We leave this study as future work for a lower memory footprint.

6 Conclusion and Future Work

We design and implement AdaPyramid, a framework to reduce as much on-device inference latency as possible, especially for high-resolution videos, while still achieving the accuracy demand approximately. The design is based on the observations of high-resolution video features. We then make extensive explorations to utilize the observations in order to optimize our system. We also discuss the rationality of every design sufficiently in this paper.

Moving forward, we will improve AdaPyramid regarding three aspects. First, the construction of the configuration pool requires further investigation. A good pool should cover a wide range of detection capabilities in a fine-grained way. To realize this goal, besides considering each existing knob (i.e., the NN's type and frame scaling ratio) more carefully, we can also add more knobs. Second, we will try to extend AdaPyramid from the current fixed camera scenarios to moving camera scenarios by adaptively updating the pyramid-like partition scheme. Lastly, we will study how to extend AdaPyramid for parallel inference on heterogeneous edge devices, considering each frame has been partitioned into different regions which have no dependency in inference amongst each other.

Acknowledgments

We thank the editor and anonymous reviewers. This work was supported in part by NSFC (62202233, 61832008), Double Innovation Plan of Jiangsu Province (JSSCBS20220409), and Collaborative Innovation Center of Novel Software Technology and Industrialization.

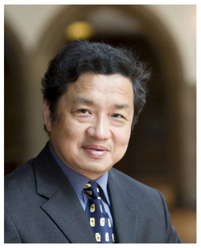
References

[1] G. Ananthanarayanan and et al., "Video analytics - killer app for edge computing," in *Proc. of MOBISYS 2019*, p. 695–696.
[2] R. Bhardwaj and et al., "Ekya: Continuous learning of video analytics models on edge compute servers," in *Proc. of USENIX NSDI 2022*.
[3] L. N. Huynh and et al., "Deepmon: Mobile gpu-based deep learning framework for continuous vision applications," in *Proc. of ACM MOBISYS 2017*, p. 82–95.
[4] M. Xu and et al., "Video analytics with zero-streaming cameras," in *Proc. of USENIX ATC 2021*.
[5] M. Xu, M. Zhu, and et al., "Deepcache: Principled cache for mobile deep vision," in *Proc. of ACM MOBICOM 2018*, p. 129–144.
[6] N. Chen and et al., "Resmap: Exploiting sparse residual feature map for accelerating cross-edge video analytics," in *Proc. of IEEE INFOCOM 2023*.
[7] "Qualcomm vision ai devkit," *https://bit.ly/328LjBF*.
[8] "Hikvision 4k camera," *https://bit.ly/2NViRiT*.
[9] L. Liu and et al., "Edge assisted real-time object detection for mobile augmented reality," in *Proc. of ACM MOBICOM 2019*.
[10] X. Wang and et al., "Edgeduet: Tiling small object detection for edge assisted autonomous mobile vision," in *Proc. of IEEE INFOCOM 2022*, pp. 1–10.
[11] K. Yang and et al., "Flexpatch: Fast and accurate object detection for on-device high-resolution live video analytics," in *Proc. of IEEE INFOCOM 2022*, pp. 1898–1907.
[12] W. Zhang and et al., "Elf: accelerate high-resolution mobile deep vision with content-aware parallel offloading," in *Proc. of ACM MOBICOM 2021*, p. 201–214.
[13] S. Jiang and et al., "Flexible high-resolution object detection on edge devices with tunable latency," in *Proc. of ACM MOBICOM 2021*, p. 559–572.
[14] G. Zhao and et al., "Collaborative training between region proposal localization and classification for domain adaptive object detection," in *Proc. of ECCV 2020*, p. 86–102.
[15] Y. Zheng and et al., "Cross-domain object detection through coarse-to-fine feature adaptation," in *Proc. of IEEE/CVF CVPR 2020*, pp. 13 766–13 775.
[16] "Common objects in context," *https://cocodataset.org*.
[17] M. Najibi and et al., "Autofocus: Efficient multi-scale inference," in *Proc. of IEEE ICCV 2019*, pp. 9745–9755.
[18] J. F. Henriques and et al., "High-speed tracking with kernelized correlation filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 3, pp. 583–596, 2015.
[19] B. D. Lucas and et al., "An iterative image registration technique with an application to stereo vision," in *Proc. of IJCAI 1981*.
[20] A. Bewley and et al., "Simple online and realtime tracking," in *Proc. of IEEE ICIP 2016*, pp. 3464–3468.
[21] N. Wojke and et al., "Simple online and realtime tracking with a deep association metric," in *Proc. of IEEE ICIP 2017*, pp. 3645–3649.
[22] C. Zhang and et al., "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proc. of ACM/SIGDA FPGA 2015*, pp. 161–170.
[23] N. P. Jouppi and et al., "In-datacenter performance analysis of a tensor processing unit," in *Proc. of ISCA 2017*, pp. 1–12.

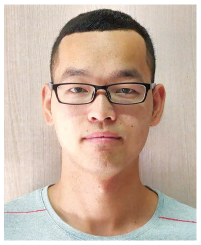
[24] B. Fang and et al., "Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision," in *Proc. of ACM MOBICOM 2018*, pp. 115–127.
[25] Y. He and et al., "Channel pruning for accelerating very deep neural networks," in *Proc. of IEEE ICCV 2017*, pp. 1398–1406.
[26] Z. He and et al., "Simultaneously optimizing weight and quantizer of ternary neural network using truncated gaussian approximation," in *Proc. of IEEE CVPR 2019*, pp. 11 430–11 438.
[27] J. Wu and et al., "Quantized convolutional neural networks for mobile devices," in *Proc. of IEEE CVPR 2016*, pp. 4820–4828.
[28] J. Yim and et al., "A gift from knowledge distillation: Fast optimization, network minimization and transfer learning," in *Proc. of IEEE CVPR 2017*, pp. 4133–4141.
[29] B. Zoph and et al., "Learning transferable architectures for scalable image recognition," in *Proc. of IEEE CVPR 2018*, pp. 8697–8710.
[30] J. Jiang and et al., "Chameleon: scalable adaptation of video analytics," in *Proc. of ACM SIGCOMM 2018*, p. 253–266.
[31] H. Zhang and et al., "Live video analytics at scale with approximation and delay-tolerance," in *Proc. of USENIX NSDI 2017*.
[32] C. Wang and et al., "Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics," in *Proc. of IEEE INFOCOM 2020*, p. 257–266.
[33] M. Zhang and et al., "Casva: Configuration-adaptive streaming for live video analytics," in *Proc. of IEEE INFOCOM 2022*.
[34] X. Shi and et al., "Osca: Online user-managed server selection and configuration adaptation for interactive mar," in *Proc. of IEEE/ACM IWQoS 2023*, pp. 1–10.
[35] "Yolov5," *https://github.com/ultralytics/yolov5*.
[36] S. Ren and et al., "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Proc. of NIPS 2015*, pp. 91–99.
[37] T.-Y. Lin and et al., "Focal loss for dense object detection," in *Proc. of IEEE ICCV 2017*, pp. 2980–2988.
[38] J. Yi and et al., "Eagleeye: Wearable camera-based person identification in crowded urban spaces," in *Proc. of ACM MOBICOM 2020*, pp. 9745–9755.
[39] M. Gao and et al., "Dynamic zoom-in network for fast object detection in large images," in *Proc. of IEEE/CVF CVPR 2018*.
[40] M. Yuan and et al., "Infi: end-to-end learnable input filter for resource-efficient mobile-centric inference," in *Proc. of ACM MOBICOM 2022*, pp. 228–241.
[41] Y. Chai, "Patchwork: A patch-wise attention network for efficient object detection and segmentation in video streams," in *Proc. of IEEE/CVF ICCV 2019*, pp. 3415–3424.
[42] M. Gao and et al., "Towards high-resolution salient object detection," in *Proc. of IEEE/CVF ICCV 2019*, pp. 7234–7243.
[43] T. Y.-H. Chen and et al., "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proc. of ACM SENSYS 2015*.
[44] K. Apicharttrisorn and et al., "Frugal following: Power thrifty object detection and tracking for mobile augmented reality," in *Proc. of ACM SENSYS 2019*, pp. 96–109.
[45] M. Liu and et al., "Continuous, real-time object detection on mobile devices without offloading," in *Proc. of IEEE ICDCS 2020*.
[46] L. Xu and et al., "Scale invariant optical flow," in *Proc. of ECCV 2012*, pp. 385–399.
[47] "Panda dataset," *http://www.panda-dataset.com*.
[48] X. Wang and et al., "Panda: A gigapixel-level human-centric video dataset," in *Proc. of IEEE/CVF CVPR 2020*, pp. 3265–3275.
[49] "Nvidia jetson agx xavier," *http://bit.ly/3nVJBM7*.
[50] A. Paszke and et al., "Pytorch: An imperative style, high-performance deep learning library," in *Proc. of NIPS 2019*.
[51] Z. Xiao and et al., "Towards performance clarity of edge video analytics," in *Proc. of IEEE/ACM SEC 2021*, p. 148–164.

Xiaohang Shi received his BS degree from the Department of Computer Science and Technology, Nanjing University, Nanjing, China, in 2020, where he is currently working towards a PhD degree under the supervision of associate professor Sheng Zhang. He is a member of the State Key Laboratory for Novel Software Technology. Currently, his research interests include edge computing and video analytics.

Sheng Zhang is an associate professor in the Department of Computer Science and Technology, Nanjing University. His research interests include cloud computing and edge computing. To date, he has published more than 80 papers, including those which appeared in JSAC, TMC, TON, MobiHoc, ICDCS, and INFOCOM. He received the Best Paper Award of IEEE ICCCN 2020 and the Best Paper Runner-Up Award of IEEE MASS 2012.



Jie Wu (F'09) is the Director of the Center for Networked Computing and Laura H. Carnell professor at Temple University. He also serves as the Director of International Affairs at College of Science and Technology. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. Dr. Wu regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Mobile Computing, IEEE Transactions on Service Computing, Journal of Parallel and Distributed Computing, and Journal of Computer Science and Technology. Dr. Wu is a CCF Distinguished Speaker and a Fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.



Ning Chen is currently pursuing a PhD degree in the Department of Computer Science and Technology, Nanjing University, under the supervision of Prof. Sheng Zhang. His research interests include edge computing, deep reinforcement learning, and video streaming. To date, he has published several papers, including those which appeared in INFOCOM, TPDS, SECON, Computer Networks, ICPADS, et al.

Ke Cheng received his bachelor's degree in Information Management and Information System with a minor degree in Computer science and Technology from Nanjing University, Nanjing, China, in 2021. He is currently working toward a PhD degree in the Department of Computer Science and Technology, Nanjing University. He is a member of the State Key Laboratory for Novel Software Technology. His research interests include edge computing and blockchain.

Yu Liang is a lecturer at Nanjing Normal University. She received the MS and PhD degrees from Nanjing University in 2011 and 2021, respectively. She was a senior software engineer in Trend Micro China Development Center between 2011 and 2017. Her research interests include edge intelligence and edge computing. Her publications include those appeared in TMC, TPDS, TON, Computer Networks, Computer Communications, IEEE ICDCS, IEEE MSN, and IEEE Globecom.

Sanglu Lu received her BS, MS, and PhD degrees from Nanjing University in 1992, 1995, and 1997, respectively, all in computer science. She is currently a professor in the Department of Computer Science and Technology and the State Key Laboratory for Novel Software Technology. Her research interests include distributed computing, wireless networks, and pervasive computing. She has published over 80 papers in refereed journals and conferences in the above areas. She is a member of IEEE.