ACCELERATING OBJECT DETECTION AND TRACKING PIPELINES FOR EFFICIENT EDGE VIDEO ANALYTICS

ACCELERATING OBJECT DETECTION AND TRACKING PIPELINES FOR EFFICIENT EDGE VIDEO ANALYTICS

By RENJIE XU, M.E.

A Thesis Submitted to the School of Graduate Studies in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

McMaster University

DOCTOR OF PHILOSOPHY (2025)

Hamilton, Ontario, Canada (Department of Computing and Software)

TITLE:	Accelerating Object Detection and Tracking Pipelines for Efficient Edge Video Analytics
AUTHOR:	Renjie Xu M.E. (Circuits and Systems), Nanjing Forestry University, Nanjing, China
SUPERVISOR:	Dr. Rong Zheng
CO-SUPERVISOR:	Dr. Saiedeh Razavi
NUMBER OF PAGES:	xx, 154

Lay Abstract

Video analytics is a technique that can extract insightful information from videos, driving real-world applications such as traffic monitoring, where rapid and accurate responses are critical for safety. However, existing video analytics pipelines are compute-intensive, making them difficult to run efficiently on resource-constrained edge devices. This thesis proposes three novel approaches that significantly accelerate video analytics without compromising accuracy. These approaches intelligently adjust how videos are analyzed by selecting appropriate resolutions and processing models, and by focusing only on the most informative parts of each frame, greatly reducing unnecessary computation and communication. Extensive experiments demonstrate that the proposed approaches enhance the trade-off between accuracy and efficiency, providing a strong foundation for efficient and reliable edge video analytics.

Abstract

Edge computing enables rapid video analytics by processing data closer to the source, thereby reducing end-to-end latency. This gives rise to the paradigm of edge video analytics (EVA). Object detection and object tracking are key building blocks of video analytics pipelines (VAPs), as their outputs directly impact the performance of downstream tasks. In real-world applications like traffic monitoring, timely and accurate responses are critical—delayed or inaccurate results can compromise safety. However, achieving such an accuracy-efficiency balance at the edge is particularly challenging due to two main factors: the compute-intensive nature of modern Convolutional Neural Network (CNN)- or Vision Transformer (ViT)-based models, and the limited computational and communication resources on edge devices.

This thesis aims to improve the efficiency of object detection and tracking pipelines without sacrificing accuracy, enabling efficient and reliable EVA. Conventional pipelines often adopt fixed configurations (e.g., frame resolution and backbone model) or process entire frames uniformly, overlooking the dynamic and spatially diverse nature of video content, resulting in considerable resource waste. To address these limitations, we propose three novel approaches: **FastTuner**, a model-agnostic framework that dynamically selects the optimal frame resolution and backbone model at runtime to accelerate multi-object tracking (MOT) pipelines; **BlockHybrid**, which leverages

a policy network to classify each frame into "hard" and "easy" blocks, and processes them with either a block-wise detector or a lightweight tracker accordingly; and **SEED**, an end-to-end framework that couples block selection with block execution, enabling unified and efficient selection and execution of informative blocks in ViT-based object detectors. Extensive evaluations across multiple datasets and deployment scenarios demonstrate the effectiveness and generality of the proposed methods. Together, these contributions pave the way for more adaptive and scalable video analytics in real-world edge environments.

To my family,

friends, and mentors.

Acknowledgements

First and foremost, I would like to express my deepest thanks to my supervisor, Dr. Rong Zheng and co-supervisor Dr. Saiedeh Razavi for their continuous support, insightful guidance, and encouragement throughout my Ph.D. studies. Their expertise, patience, and high standards have greatly shaped both my research and personal growth. I feel truly fortunate to have the opportunity to work under their supervision.

I am also very grateful to my committee members: Dr. Wenbo He, Dr. Douglas Down, for their valuable feedback and thoughtful suggestions, which helped improve the quality of my work. Their support throughout the various stages of this thesis has been instrumental.

Special thanks go to my master's supervisors, Dr. Yunfei Liu and Dr. Haifeng Lin, who inspired my interest in research and provided me with a solid foundation during my early academic journey. Their guidance has played a vital role in preparing me for doctoral studies, and I will always appreciate the mentorship they provided.

I would like to thank all members of the WiSeR group for creating such a friendly and collaborative research environment. I am especially grateful to Dr. Keivan Nalaie, my best friend here. Our regular discussions related to both academic and everyday life have not only helped me improve my research but also made my time here more enjoyable.

I also want to express my sincere appreciation to my close friend, Mr. Xuli Cai, for always being there to talk and listen. His companionship and support have helped me get through stressful times and kept me balanced along the way.

Last but not least, I owe my deepest gratitude to my parents for their unconditional love, understanding, and constant support. Their belief in me has been my greatest source of strength, and this work would not have been possible without them. I am equally thankful to my home country, China, and the China Scholarship Council for providing the financial support that enabled me to pursue my doctoral studies abroad.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

VA	Video Analytics
EVA	Edge Video Analytics
DL	Deep Learning
RL	Reinforcement Learning
CV	Computer Vision
AI	Artificial Intelligence
IoT	Internet of Things
WAN	Wide Area Network
BS	Base Station
QoE	Quality of Experience
VAP	Video Analytics Pipeline
ViT	Vision Transformer
CNN	Convolutional Neural Network

MOT	Multi-Object Tracking
HoC	Histogram of Color
HOG	Histogram of Oriented Gradients
DNN	Deep Neural Network
MLP	Feed-Forward Network or Multilayer Perceptron
MOTA	Multi-Object Tracking Accuracy
mAP	mean Average Precision
FP	False Postive
FN	False Negative
IDSW	Identity Switch
GT	Ground Truth
bbox	bounding box
IoU	Intersection over Union
re-ID	re-identification
FCN	Fully Convolution Network
SOTA	State of the Art
DLA	Deep Layer Aggregation
MAC	Multiply-Accumulate Operation

FR	Full-Resolution
ITS	Intelligent Transportation System
RSU	Roadside Unit
V2X	Vehicle-to-Everything
IB	Informative Block
non-IB	non-Iormative Block
HB	Hard Block
IG	Information Gain
TE	Task Error
RTT	Round-Trip Time
TCP	Transmission Control Protocol
FPS	Frames Per Second
FD	Full-Frame Detector
MSA	Multi-Head Self-Attention
LN	Layer Normalization
HFR	Hybrid Feature Reconstruction
FPN	Feature Pyramid Network
QP	Quantization Parameter

RTMP	Real-Time Messaging Protocol
RTSP	Real-Time Streaming Protocol
WebRTC	Web Real-Time Communication
ML	Machine Learning
SIFT	Scale-Invariant Feature Transform
SVM	Support Vector Machine
k-NN	k-Nearest Neighbor
RoI	Region of Interest
RCNN	Region-Based Convolutional Neural Network
RPN	Region Proposal Network

Declaration of Academic Achievement

The research presented in this thesis was conducted by the author over the period 2021–2025. The author was the primary contributor, responsible for formulating the research problem, designing and implementing the proposed methods, performing experiments, and drafting the manuscripts.

Chapter 1

Introduction

1.1 Motivation

Cameras are in every corner of our cities in this information-centric era. According to [1], one surveillance camera is installed for every eight people on the planet nowadays, with mature markets (e.g., China and the United States) having one camera for every four people. Such explosive video data is beyond human capacity to make sense of what is happening manually. Video analytics (VA) aims to automatically and efficiently recognize objects and identify interesting events in unstructured video data. It can drive a large number of applications with wide-ranging impacts on our society. Examples of such applications include security surveillance in public and private venues, assisted and autonomous driving and consumer applications such as digital assistants for real-time decision-making [2].

Early-stage video analytics is based on conventional image processing techniques, which mainly rely on human expertise and empirical knowledge, and thus are not robust to changes in lighting conditions, viewing angles, weather conditions, etc. [2]. Deep learning (DL) has made striking breakthroughs in many fields, especially in computer vision (CV). Advanced CV technologies, e.g., object classification, detection, and tracking, enable extracting more accurate information and insights from video feeds. The resulting insights can help people make smarter and faster decisions.

However, many DL-driven applications are compute-intensive, thus not friendly to resource-constrained Internet-of-Things (IoT) devices. The conventional wisdom is to offload all workloads from devices to a cloud via wide area networks (WANs), where powerful data centers are located. This computing paradigm, known as cloud computing, suffers from high service delays due to long geographical distances and potential network congestion. According to [3], worldwide data will reach 175 zettabytes (ZB)

by 2025, 51% of which will be created by IoT devices. Digesting such massive data in the cloud incurs excessive delays, making such solution inadequate for mission-critical applications, e.g., security surveillance [4] and autonomous driving [5], where delayed responses can compromise safety.

Edge computing, an emerging computing paradigm, has recently been recognized as a viable alternative to cloud computing. It is a distributed architecture that reduces latency by hosting applications and computing resources at locations geographically closer to the data source. Simply put, edge computing alleviates data transfer latency by processing data on local edge nodes rather than in a remote cloud. An edge node can vary in size and capability, ranging from tiny processing units co-located with IoT devices, to IT infrastructures in close proximity to base stations (BSs). These nodes, distributed at the network edge, can significantly alleviate the workloads and traffic congestions of the cloud, thereby reducing the service delay and improving the quality of experience (QoE) of users.

Edge computing is an extension of cloud computing by pushing centralized workloads to the network edge. Instead of entirely relying on the cloud, edge computing, a flexible computing paradigm leveraging both edge and cloud capabilities effectively, is gaining traction in building VA systems [6]. Therefore, we are now witnessing the convergence of video analytics and edge computing, namely, *edge video analytics* (EVA). Many techniques have been proposed to improve the efficiency of EVA.

Configuration Optimization. A typical video analytics pipeline (VAP) consists of multiple processing components, among which core modules such as object detectors often expose several tunable parameters, referred to as *knobs*. A knob, such as model choice or input resolution, offers trade-offs between computational cost and

accuracy. A *configuration* represents a particular combination of these knobs. Importantly, no single configuration consistently delivers the best performance across all deployment scenarios [7]. Therefore, the selection of an appropriate configuration plays a crucial role in determining the accuracy-latency trade-off of a VAP [8, 7, 9, 10]. For example, using high-resolution input or a deeper neural network may improve detection accuracy but also increase latency and resource consumption. A configuration is considered *Pareto-optimal* if it is impossible to improve one metric (e.g., accuracy) without degrading another (e.g., latency). Selecting the best configuration is non-trivial as optimal choices can vary over time due to changes in video content, resource availability, or application requirements. Existing approaches rely on separate modules, such as offline or online profiling, to select the best configuration [10, 7, 9]. While offline profiling is efficient, it cannot adapt to dynamic video content. Online profiling offers adaptability but adds computational overhead. In both cases, configuration selection is decoupled from the core task (e.g., detection, tracking), limiting overall efficiency.

Conditional Execution. Conditional execution, also known as *dynamic or selective execution*, aims to reduce redundant computation by adapting the processing strategy based on the input content. Early methods focused on coarse-grained, frame-level decisions, such as frame skipping [11, 12], resolution scaling [13–15], or early exit [16–18], where entire frames could be dropped, downsampled, or terminated early in the processing pipeline depending on their perceived importance. These strategies helped avoid unnecessary computation on static or uninformative frames. Recent research has extended this concept to finer-grained, block-wise execution [19, 20]. Instead of treating a frame as a whole, these approaches selectively process only the

spatial regions within a frame that are likely to be informative, such as blocks containing target objects. By dynamically adjusting the computation granularity within a frame, these methods further enhance efficiency while preserving task accuracy. Such techniques are particularly valuable in edge environments, where computational and communication resources are limited. However, developing an effective block-wise conditional execution solution remains challenging. It requires accurate identification of informative regions under diverse scene conditions, as well as efficient support for selective execution at the block level. Furthermore, processing frames in a non-uniform manner can lead to *block artifacts*, where inconsistencies between features from processed and unprocessed (or lightly processed) blocks degrade overall task performance. Another major challenge is that if block selection and block execution are done independently, performance degradation may occur, as the selection is made without awareness of how the selected blocks will ultimately impact the execution results.

Accelerating ViT-based VAPs. Vision Transformers (ViTs) have recently demonstrated superior performance over Convolutional Neural Networks (CNNs) in a wide range of computer vision tasks, including object detection, semantic segmentation, and video understanding [21]. Their ability to model long-range dependencies and global context makes them attractive for complex video analytics. As a result, ViT-based architectures have been increasingly adopted in VAPs. However, the high computational complexity and memory footprint of ViTs pose significant challenges for deployment on edge devices with limited resources. Unlike CNNs, which benefit from local receptive fields and weight sharing, ViTs rely heavily on self-attention mechanisms with quadratic complexity in sequence length, making them less efficient

in processing high-resolution video inputs. To address these challenges, recent research has explored techniques such as layer pruning [22], network quantization [23], and efficient attention mechanisms [24, 25] to reduce the inference cost of ViTs. While these techniques improve efficiency, they are typically *context-agnostic*, applying uniform optimization strategies regardless of input content. This limits their ability to adapt to spatial and semantic variations in real-world videos. In contrast, block-wise conditional execution enables fine-grained, *context-aware* processing by selectively activating computation only on informative blocks, making it a promising direction for efficient ViT-based video analytics.

Distributed Processing. Conventional VAPs often adopt a simple design, where entire frames are captured by a camera and transmitted one by one to a server for processing [26]. This approach ignores the structural characteristics of the pipeline and leads to unnecessary communication and computation overhead, especially when large portions of the video contain uninformative content. Distributed processing in video analytics refers to splitting the workload across multiple devices, such as cameras, edge nodes, and cloud servers, to meet real-time and resource constraints. This approach enables early filtering or lightweight analysis on resource-constrained devices (e.g., smart cameras), while offloading heavier computation (e.g., object detection or tracking) to more powerful nodes [27–31]. Effective partitioning and scheduling of tasks in a distributed VAP are critical for minimizing end-to-end latency and preserving bandwidth. However, fully exploiting the benefits of distributed processing requires pipeline-aware optimizations. In the context of block-wise conditional execution, for instance, block selection can be performed on the camera side, and only the selected blocks are transmitted for further processing. This fine-grained data flow

not only reduces transmission cost but also preserves task accuracy.

1.2 Contributions

This thesis, as summarized in Figure 1.1, contributes to the acceleration of VAPs on resource-constrained edge platforms. It centers around two key directions: configuration optimization and block-wise conditional execution.

Configuration Optimization – In Chapter 3, we present **FastTuner**, a framework designed to optimize the configuration of multi-object tracking (MOT) pipelines at runtime. At runtime, FastTuner periodically selects the best resolution and backbone model based on the input frame, and applies the selected configuration to perform object tracking. The key novelty of FastTuner includes:

- An efficient estimator that predicts the performance of different resolution and model choices at one shot.
- A model-agnostic framework that unifies configuration selection and object tracking within a shared model, eliminating the need for costly online profiling.
- Two distributed processing schemes that leverage FastTuner's adaptability to reduce both computation and communication overhead, enabling efficient deployment on heterogeneous end-edge architectures.

Block-Wise Conditional Execution – The second thread of the thesis focuses on reducing local spatial redundancy by selectively executing only the informative blocks within each frame. We have developed two different approaches toward this goal:

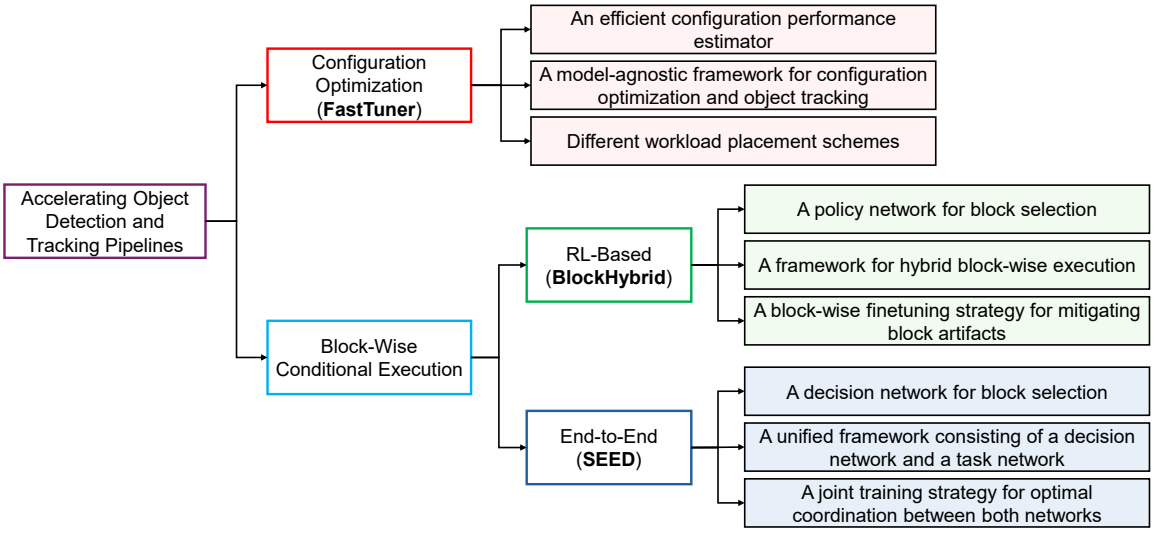


Figure 1.1: Overview of the contributions.

1) BlockHybrid (Chapter 4) is a framework that accelerates object detection pipelines by hybrid block-wise executions. Its design decouples block selection from execution, enabling a modular and flexible design compatible with both CNN- and ViT-based detectors. The main contributions of BlockHybrid are as follows:

- A policy network trained via reinforcement learning (RL) that classifies image blocks as "easy" or "hard" in each frame.
- A block-wise conditional execution framework that handles both types of blocks: hard blocks are processed by a customized block-wise detector, while easy blocks are handled by a lightweight tracker that propagates historical object information across frames, effectively reducing redundant computation.
- A block-wise fine-tuning strategy that adapts the detector to non-uniform block-wise inputs, mitigating accuracy degradation caused by block artifacts.

2) SEED (Chapter 5) proposes a fully end-to-end ViT-based selective execution

framework that tightly couples block selection with execution. Unlike BlockHybrid, which trains the decision module and detector separately, SEED unifies them within a single training and inference pipeline. The key contributions of SEED include:

- A lightweight decision network that identifies semantically informative blocks based on the input.
- An end-to-end framework that unifies a decision network and a block-wise detector, with its generalizability demonstrated through two variants employing different selective execution strategies: SEED-TR (token reuse) and SEED-EE (early exit), each offering a distinct trade-off between accuracy and efficiency.
- A multi-stage training strategy that jointly optimizes both networks to ensure tight coordination between block selection and execution, improving overall efficiency while mitigating block artifacts.

1.3 Organization

The main technical content of this sandwich thesis comprises two published journal papers and one paper currently under review. The remainder of the thesis is organized as follows:

- **Chapter 2** provides an overview of VAP preliminaries, related work in accelerating VAPs, performance metrics, commonly used datasets and hardware.
- **Chapter 3** presents **FastTuner**, a configuration optimization framework that adaptively selects the best resolution and model variant at runtime to accelerate MOT under dynamic video content.

- **Chapter 4** describes **BlockHybrid**, a RL-based block-wise conditional execution framework that accelerates CNN- and ViT-based object detection pipelines through decoupled block selection and execution.
- **Chapter 5** introduces **SEED**, an end-to-end selective execution framework that speeds up ViT-based object detection pipelines through unified block selection and execution.
- **Chapter 6** concludes the thesis by summarizing key insights, highlighting current limitations, and discussing potential directions for future work.

Chapter 2

Background

2.1 Preliminaries of Video Analytics Pipeline

Video analytics, also known as video content analysis, refers to the process of automatically extracting valuable information and insights from video data using techniques such as CV and DL. It involves recognizing patterns, detecting objects and tracking movements in order to analyze, interpret, and understand video content and make data-driven decisions. A VAP refers to a series of sequential steps or stages through which the data passes to be transformed, analyzed, and processed in a VA application. The pipeline represents the overall flow and organization of the various operations and algorithms applied to the data, from the initial input to the final output. Each stage in the pipeline typically focuses on a specific task or function, and the output from one stage becomes the input for the next stage, allowing for a modular and structured approach to processing the data.

Typically, a VAP is composed of multiple video processing modules, which can vary across applications, as shown in Figure 2.1,

Frame Encoding and Decoding: Frame encoding reduces communication overhead by compressing video frames before transmission [32]. In existing VAPs, encoding is typically applied either on a per-frame basis or over multiple consecutive frames grouped into a segment, depending on the application's latency requirements [26]. For latency-sensitive applications such as intelligent transportation systems (ITS) [33], encoding and transmitting frames individually ensures minimal delay. In contrast, applications with more relaxed latency constraints may batch frames into segments for higher compression efficiency. Encoded data is often transmitted via live streaming protocols such as real-time messaging protocol (RTMP), real-time streaming protocol (RTSP), and web real-time communication (WebRTC). At the edge or cloud, frame

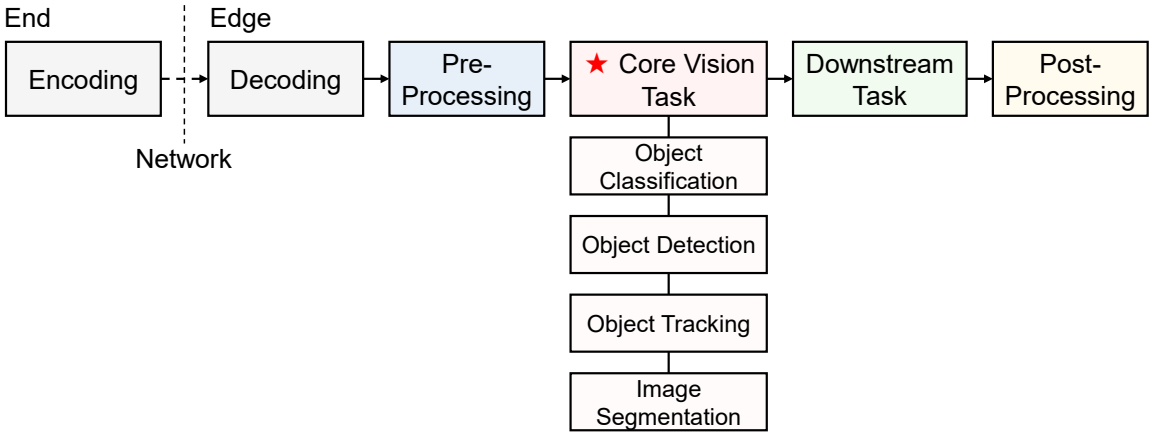


Figure 2.1: Components of a video analytics pipeline.

decoding is performed to reconstruct the original content for downstream processing in VAPs.

Pre-processing: After decoding, frames are pre-processed before being subject to further analytics. The pre-processing operations include image resizing [34], cropping [35], super-resolution [36], denoising [37, 36, 38], deblurring [38], dehazing [39], and deraining [38]. In general, these operations aim to improve the view quality and can therefore benefit the subsequent procedures. In real-world applications, a video may contain multiple sources of noise. For instance, traffic videos captured at night may suffer from poor illumination and motion blur. OpenCV [40], a well-known library in image and video processing, implements a wide range of pre-processing algorithms for image denoising, resizing, rotating, padding, normalization, color space conversions, morphological operations, background subtraction for video motion detection, etc.

Core Vision Task: In VAPs, the core vision task is the most important component, as its output directly feeds into subsequent stages. Consequently, the performance of the core task has a significant impact on the quality of downstream tasks

and the overall application. Recent research has primarily focused on optimizing the accuracy-efficiency trade-off of the core task [2]. Common core tasks include object classification, detection, tracking, and segmentation, all of which are classical vision tasks in CV. In this thesis, we mainly focus on object detection and tracking tasks.

- *Object Classification*: Object classification, also known as object recognition or object identification, maps an object into one of a finite set of classes. Early object classification is primarily based on handcrafted features and shallow machine learning (ML) models. Methods such as Scale-Invariant Feature Transform (SIFT) and Histogram of Oriented Gradients (HOG) are used to extract features from images, which are then fed into classifiers like Support Vector Machines (SVM) or k-Nearest Neighbors (k-NN) [41, 42]. These methods, while effective for certain scenarios, often struggle with variations in lighting, pose, and scale. With the advent of DL, the paradigm shifted towards end-to-end learning. CNN-based classifiers, such as ResNet [43] and MobileNet [44], have become dominating approaches, leveraging large labeled datasets to predict the class of target objects with remarkable accuracy, surpassing traditional ML-based approaches.

- *Object Detection*: Object detection involves locating and recognizing objects in frames. Traditional object detection methods first identify regions of interest (RoIs), using image processing methods like background subtraction [45], frame differencing [46], and optical flow [47, 48]. Once these regions are detected, they are classified using an object classifier. Nowadays, object detection algorithms typically leverage Deep Neural Networks (DNNs) to achieve high accuracy and can be classified into two categories: two-stage and one-stage [49]. A two-stage

object detector first extracts RoIs, and then makes a separate prediction for each of these regions. Faster region-based convolutional neural network (RCNN) [50] represents a classical two-stage detector. It employs a region proposal network (RPN) to generate region proposals and performs classification on these regions separately. A one-stage object detector, in contrast, simply applies a single DNN model for both object localization and recognition. The two tasks are cast as a unified regression problem. The most widely-known one-stage detectors include the YOLO family [51–53] and the SSD family [54]. In general, one-stage detectors are much faster but less accurate than two-stage ones.

- *Object Tracking*: Object tracking is the process of locating objects and estimating their trajectories from a video sequence. Conventional methods follow the tracking-by-detection paradigm [55], performing tracking sequentially using two separate models [56, 57]. A detector first detects bounding boxes (bboxes) of objects in each frame, after which a re-identification (re-ID) model extracts visual features from each bounding box and links the objects based on these features and motion cues. Such two-stage methods, while effective, are computationally intensive, especially when the scene is crowded. Recent advancements in multi-task learning have led to joint models where detection and re-ID tasks share a common backbone, significantly reducing inference time. These integrated models are often termed one-shot trackers [58–61].

- *Image Segmentation*: Image segmentation involves partitioning an image into multiple segments, each representing a distinct object or region. It simplifies and changes the image representation into something more meaningful and easier to analyze. Among the various types of image segmentation, semantic

segmentation assigns each pixel to a specific class, while instance segmentation classifies each pixel and differentiates between distinct instances of the same object class [62]. Panoptic segmentation, on the other hand, unifies the tasks of semantic and instance segmentation, producing coherent labelling of all pixels, considering both regions and objects [62]. These techniques can be integrated with other CV tasks, such as object detection and tracking, to achieve a more comprehensive understanding of scenes.

Downstream Task: Downstream tasks in a VAP build upon the structured outputs generated by the core task to enable more application-specific analysis and decision-making [2]. These tasks vary depending on the context, and may include behavior analysis, trajectory forecasting, event detection, anomaly detection, or scene understanding. For example, in traffic monitoring, downstream modules may use object trajectories (generated by a tracking module) to predict vehicle intent or detect potential collisions. In retail analytics, person detection and tracking results may feed into customer flow modeling or dwell time estimation. Since downstream tasks rely heavily on the correctness and timeliness of core task outputs, any degradation in core task performance can lead to cascading errors and reduced system utility. Therefore, efficient and accurate execution of the core task is critical to the robustness of the entire pipeline.

Post-Processing: Post-processing at the final stage of a VAP refines and integrates the outputs of preceding tasks, ensuring that the pipeline delivers coherent and reliable application-level decisions [2]. In some cases, post-processing also involves fusing outputs from multiple downstream tasks to generate a unified interpretation of the scene [63]. For example, in traffic monitoring, outputs from object tracking and

event detection may be combined to identify high-risk scenarios such as illegal lane changes or near collisions. As such, effective post-processing is essential for delivering accurate, stable, and actionable insights to end users.

2.2 Related Work

This section reviews prior research that closely relates to the core contributions of this thesis. We focus on three key areas: configuration optimization in VAPs, block-wise conditional execution for efficient inference, and the acceleration of ViTs.

2.2.1 Configuration Optimization in VAPs

In VAPs, configurations involve various tunable knobs: video quality (e.g., resolution, frame rate, bitrate) [64–71], DNNs [64, 66, 14, 72, 73], resource allocations (e.g., CPU cores, network bandwidth) [64, 66, 14, 74], camera parameters (e.g., brightness, contrast, colorfulness, orientations) [75, 76], etc. Different configurations can yield distinctive accuracy-latency trade-offs. Choosing a good configuration that maximizes execution efficiency while maintaining analytics quality can be accomplished by first gathering information regarding execution characteristics, also known as profiles [77]. To acquire an accurate profile, one can conduct a one-time but exhaustive, offline profiling. This process involves a profiler executing the pipeline under various configurations and inputs, and outputting accuracy and latency measurements. The inputs (i.e., frames) used in the profiling need to be representative of target application scenarios. Profiling costs can be prohibitive since the configuration space tends

to grow exponentially with the number of knobs and their respective values. For instance, VideoEdge [7] considers a configuration space of 1800 combinations stemming from five knobs. VideoStorm [9], on the other hand, needs to profile 414 configurations, requiring 20 CPU-days using a 10-minute video. Even though parallelism has been exploited to accelerate profiling [9, 78], the high resource demand for exhaustive profiling remains a challenge. To alleviate this problem, VideoEdge [7] merges common components among multiple configurations and caches intermediate results to avoid redundant executions. Another way to reduce profiling costs is to prune the configuration space. ApproxDet [79] only profiles 20% of the configurations, at the cost of generating a less accurate profile [78].

However, the static nature of offline profiling can lead to less relevant decisions when scene changes are not reflected in profiled inputs. Moreover, video content can vary greatly over time, which means that a configuration that is currently optimal may lose its effectiveness in the future. To address these limitations, online profiling conducts continuous profiling in a live environment, adapting to the temporal variability of video content. Different from offline profiling, which is done once or infrequently (e.g., once a day [10]), online profiling updates the profile periodically (e.g., every few seconds or minutes [10]) during video streaming [2]. The main challenge of online profiling lies in minimizing the overhead of periodic profiling, which, as stated earlier, is substantial even when done once. To mitigate this challenge, Chameleon [10] takes a two-step approach to perform online profiling. Initially, it conducts an exhaustive online profiling to profile all configurations, resulting in several candidate configurations. Then, it exploits cross-camera correlation and video content consistency to distribute and propagate these candidates, both spatially and temporally,

to amortize the cost of full profiling. It further reduces the configuration space by exploiting independence among some knobs [10]. AWStream [78] combines offline and online profiling. The initial offline profile is gradually refined via online profiling. For efficiency, AWStream profiles only a subset of configurations, specifically those that are Pareto-optimal (i.e., those on the Pareto frontier). A full profiling to update the current profile is triggered only when additional resources become available.

Another line of work [75, 80, 15] utilizes specific feature extractors to map an image to its feature representations. For example, LiteReconfig [80] and SmartAdapt [15] use object average size, histograms of color (HoC), HOG, and feature embeddings from a DNN feature extractor to characterize image content. These features are then used to train a separate DNN that predicts the accuracy of a given configuration. While the prediction process is efficient due to the lightweight nature of the accuracy predictor, the feature extraction step can incur substantial overhead, as it involves running multiple algorithms.

In this thesis, we focus on two knobs: resolution and backbone model, since these are applicable across nearly all VAPs. Additionally, these two knobs have a significant impact on the accuracy-latency trade-off. While other knobs (e.g., frame rate [65, 67, 81, 82, 32, 12]) are also important, they are orthogonal to our approach and therefore excluded from the configuration space considered in FastTuner. Within this context, FastTuner has several advantages compared to the aforementioned works. Firstly, it eliminates the need of a separate online profiling step, which usually requires significant time to perform exhaustively or partially at runtime. Instead, FastTuner employs DNNs to learn the heatmap representations of different configurations offline and then uses such knowledge to guide the decision making online. The performance of

different configurations can be obtained in one shot. Secondly, FastTuner integrates the core task (i.e., the MOT task) with the sub-task (i.e., configuration decision making) by sharing the backbone model rather than introducing a separate model like [79, 16, 80, 15], further improving the end-to-end latency. Finally, FastTuner exposes parameters for users to control desired accuracy-latency trade-offs for specific application scenarios.

2.2.2 Block-Wise Conditional Execution

The key idea of conditional execution is to dynamically adjust the processing strategy based on the input. This technique is initially applied at frame level. Frame sampling [11, 12, 83–88] processes only a subset of frames, skipping irrelevant or less informative ones. Adaptive resolution [13–15] dynamically adjusts frame resolution based on scene complexity, reducing resolution for simpler frames while maintaining high resolution for more challenging frames. Early-exit methods [16–18, 89–91] incorporate multiple exit points within a neural network, enabling simpler inputs to exit at shallow layers while routing more complex ones through deeper layers for thorough processing. While these approaches reduce resource consumption, they operate at frame granularity and fail to adapt to the fine-grained variations within individual frames.

Block-wise conditional execution adopts this principle by dividing frames into smaller blocks, classifying them based on their complexity or importance, performing distinct computations on each block type, and finally merging the results into a cohesive output [92–95]. The works along this line can be divided into two categories based on whether the block size is 1) non-uniform or 2) uniform. FDDIA [94] and

EHCI [95] are two representative works in the first category. They identify the informative blocks by expanding the regions around the detection results from the previous frame. The selected blocks are cropped from the input frame and sent to the server. Since these blocks are non-uniform in size, a rectangle-packing algorithm is applied to merge them into a compact frame, on which frame-wise detection is performed. The detection results are finally mapped back to the original frame. As the SOTA work in the second category, BlockCopy [19] employs a policy network to determine informative blocks based on the motion information between two consecutive frames. Due to their uniform sizes, the blocks are batch-processed by a specialized detector designed to handle blocks as inputs and produce corresponding features [20]. For uninformative blocks, their features are directly copied from the previous execution. Finally, the features from all blocks are merged into complete feature maps to generate bboxes. All the feature-level operations are enabled by customized CUDA operators. These operators can be directly applied to standard CNN-based detectors, enabling them to perform block-wise detection tasks without additional modifications. ViTs naturally support block-wise execution, as they are designed to handle variable-length patch sequences. Arena [96] partitions frames into uniform patches and performs full-frame processing at regular intervals. The tokens of these patches are cached for reuse. In the intermediate frames, informative patches are sampled from the expanded regions of detection results from the previous frame. Only the selected patches pass through a ViT encoder for self-attention computation. The resulting tokens are then merged with the cached tokens to reconstruct the complete token sequence via a single-layer decoder. Finally, the reconstructed token sequence is fed into a detection head to generate bboxes.

BlockCopy and Arena both fail to further distinguish between hard and easy blocks. Instead, they process all informative blocks with the same detector. As a result, significant computational resources are wasted by running heavy models on easy blocks, thereby missing the opportunity to further accelerate the pipeline. In BlockHybrid, we propose a novel approach, called *block-hybrid conditional execution*, which goes beyond the conventional definition of informative blocks by further differentiating blocks into hard and easy categories. Hard blocks are processed by a detector, while easy blocks are handled by a lightweight tracker. In this way, BlockHybrid dynamically allocates computation resources based on block difficulty, reducing unnecessary overhead while maintaining accuracy. Another limitation of prior works is the decoupling of block selection and execution. BlockCopy relies on a policy network that is not jointly optimized with the detector, which may result in block artifacts and compromise accuracy. EHCI, FDDIA, and Arena, on the other hand, determine informative blocks by enlarging regions around past bboxes to ensure coverage. While this strategy helps retain target objects, it often results in redundant computation, particularly in cluttered scenes. Moreover, although block-wise fine-tuning [96] improves the detector's robustness to block artifacts, their block selection remains simplistic and decoupled from the detector, leading to a suboptimal trade-off between accuracy and efficiency. To address this challenge, we propose SEED, a lightweight, context-aware, and end-to-end trainable framework that can judiciously select blocks within each frame while minimizing redundant computation. SEED jointly optimizes block selection and selective execution, offering improved efficiency without sacrificing accuracy.

2.2.3 Vision Transformer Acceleration

Many techniques have been proposed to accelerate ViTs, motivated by their high computational cost. One line of work improves the efficiency of self-attention. Standard self-attention suffers from quadratic complexity with respect to the token count, making it a computational bottleneck in ViTs, especially for long-sequence inputs. To alleviate this, several methods [97–99] approximate the attention module to reduce its complexity to linear time, significantly improving efficiency while maintaining competitive performance.

Another line of work focuses on reducing the number of tokens processed during inference. Token pruning aims to discard less informative tokens to reduce computation. DynamicViT [100] introduces a learnable module that dynamically prunes tokens based on their importance, primarily for classification tasks. SViT [101] extended this idea to dense tasks such as object detection and segmentation. It preserves pruned tokens in the feature maps and optionally reactivates them later, which proves essential for maintaining performance. It also adopts input-adaptive pruning rates and shows that lightweight selectors, such as a 2-layer feed-forward network (FFN or MLP), suffice for effective token selection. Token merging takes a different approach by retaining all information and merging similar tokens instead of discarding them. ToMe [102] progressively merges redundant tokens based on feature similarity in a lightweight manner. Token summarization further improves upon merging by selecting a few representative tokens that summarize global context. Instead of relying on similarity or attention patterns, GTP [103] learns task-specific summarization strategies and performs better than pure pruning or merging in scenarios requiring long-range contextual reasoning.

These techniques are orthogonal to SEED and can be incorporated to further boost efficiency. While SEED reduces spatial redundancy by selective block execution, token reduction and attention optimization operate at the feature and attention levels, making these techniques complementary to each other.

2.3 Datasets

The vision tasks addressed in this thesis primarily include multi-object tracking and object detection. Accordingly, we adopt the most widely used benchmark datasets in these domains to evaluate the proposed methods, as shown in Table 2.1. These datasets offer diverse scenes and realistic motion patterns, making them well-suited for assessing the accuracy and robustness of the approaches.

- **MOT17** [104]: The MOT17 dataset is a popular benchmark for evaluating MOT methods. It consists of 14 video sequences taken from various real-world scenarios, including pedestrians in a busy city environment. The dataset includes static and moving cameras, capturing both indoor and outdoor environments with diverse viewpoints (low, medium and high), lighting conditions (from daylight to nighttime), and weather conditions (sunny, cloudy, etc.). It also contains various occlusion levels, where pedestrians are partially or fully obscured by objects or other individuals, providing a comprehensive evaluation environment for tracking algorithms. MOT17 can also be adapted for evaluating object detection methods, as it provides bbox annotations for each frame.

- **WildTrack** [105]: WildTrack is a multi-camera pedestrian detection benchmark designed for evaluating advanced computer vision algorithms in real-world

scenarios. It consists of synchronized video footage captured from 7 calibrated cameras covering a public square. Since the official bounding box annotation is provided at only 5 FPS, we manually decode the raw videos and use YoloV8 [106] to generate annotations at 30 FPS. For each camera, we use the first 2000 frames.

- **ImageNet** [107]: ImageNet is a large-scale image classification dataset containing over 1 million labeled images across 1,000 categories. It is widely used for pretraining backbone networks in computer vision. In this thesis, we use ImageNet-pretrained weights to initialize the backbone of our detection models to accelerate convergence and improve downstream performance.

- **Microsoft COCO** [108]: The Microsoft COCO dataset is a large-scale benchmark for object detection, segmentation, and captioning. It contains over 150,000 labeled images across 80 object categories with rich contextual diversity. In this thesis, COCO is used to pretrain the entire model to provide a strong initialization before fine-tuning on downstream datasets such as MOT17 and WildTrack.

2.4 Performance Metrics

The performance of the proposed methods is evaluated from two key perspectives: accuracy and efficiency. For accuracy, we adopt Multiple Object Tracking Accuracy (MOTA) and mean Average Precision (mAP), which are standard metrics for evaluating multi-object tracking and object detection, respectively [2, 109, 110, 96]. For

Figure 2.2: MOT17 dataset.

Figure 2.3: Wildtrack dataset.

efficiency, we measure network traffic and end-to-end latency, capturing the communication and runtime overhead critical to edge deployment.

- **Multiple Object Tracking Accuracy**: MOTA is defined as:

$$\text{MOTA} = 1 - \frac{\sum_t \text{FP}_t + \text{FN}_t + \text{IDSW}_t}{\sum_t \text{GT}_t}, \tag{2.4.1}$$

where t is the frame index, FP_t, FN_t and IDSW_t stand for false positives (FPs), false negatives (FNs), and identity switches (IDSWs) and GT is the number

of ground truth objects. MOTA ranges from $-\infty$ to 1, where a higher score signifies better tracking performance.

- **Mean Average Precision**: mAP@0.5 refers to mean Average Precision at an Intersection over Union (IoU) threshold of 0.5. A detection is considered correct if its IoU with a ground-truth bbox exceeds 0.5. This metric assesses the model's ability to correctly localize and identify objects.

- **Network Traffic**: While communication may not always be the primary bottleneck, minimizing data transmission overhead remains essential in edge deployments, where bandwidth is often constrained. We measure network traffic as the average data size per transmission. To ensure consistency across datasets with different average data sizes, all reported values are normalized by the average full-frame size of the corresponding dataset.

- **End-to-End Latency**: This metric is critical for time-sensitive applications that require rapid response. It measures the total time elapsed from the acquisition of a video frame on the camera to the completion of its processing on the server. This process may include data compression and decompression, data transmission, model inference, and post-processing.

2.5 Hardware

Edge video analytics often runs on resource-constrained hardware platforms that must balance computation and communication overheads. NVIDIA Jetson platforms are among the most widely used hardware solutions for such applications, offering integrated CPUs and GPUs optimized for edge artificial intelligence (AI) workloads.

In this thesis, we adopt a hardware platform consisting of an NVIDIA Jetson TX2, a Dell XPS 8950 desktop and a high-performance server (called Turing server). The Jetson TX2 runs JetPack 4.6.1 and represents a low-end device commonly used in edge computing [111–113]. The desktop is equipped with an Intel i7-12700 CPU, 16GB RAM, and an NVIDIA GTX 3060 GPU, running Ubuntu 20.04, PyTorch 1.9, and CUDA 11.1. The server is equipped with an Intel Xeon E5-2620 v4 CPU, 64GB RAM, and four NVIDIA Tesla P100 GPUs (one utilized), operating on Red Hat Enterprise Linux Server (RHEL) 7.9 with PyTorch 1.7 and CUDA 11.4. All devices are connected via a 2.4GHz Wi-Fi network, provided either by a D-Link AX4800 router or the campus wireless infrastructure, emulating a realistic end-edge deployment. A snapshot of the hardware platform is shown in Figure 2.4. The specifications of these devices are reported in Table 2.2.

- **Benchmark Evaluation:** All computations are executed locally on the desktop to measure computational efficiency under controlled conditions. This setting allows us to compare the raw performance of different methods without interference from network or system heterogeneity.

- **Testbed Evaluation:** This setting reflects a practical edge deployment scenario, where the VAP is distributed between the Jetson TX2 (serving as the smart camera), and either the desktop or the server (serving as the edge server). The camera is responsible for early-stage processing such as frame acquisition, resizing, and lightweight inference, while the edge server handles more compute-intensive tasks. This setup enables an end-to-end evaluation of the pipeline in a realistic edge environment.

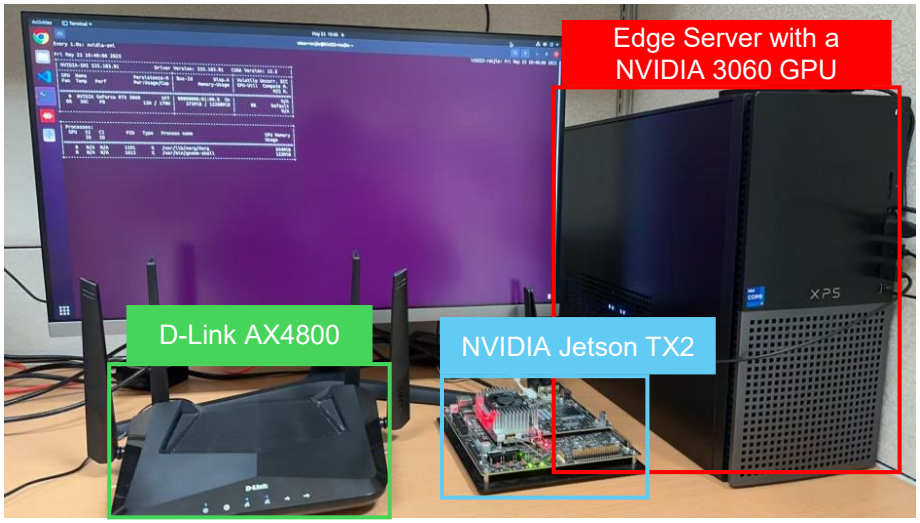


Figure 2.4: Snapshot of the hardware platform used in this thesis.

Table 2.1: Details of the datasets

Dataset	Resolution	FPS	Sequences	Boxes	Frames	Camera	Condition
MOT17	1920×1080, 640×480	30,25,14	14	300,373	11,235	static,moving	indoor,outdoor
WildTrack	1920×1080	60	7	440,800	14,000	static	outdoor
ImageNet	variable	–	–	–	1,431,167	–	indoor,outdoor
COCO	variable	–	–	886,284	163,957	–	indoor,outdoor

Table 2.2: Specifications of the devices

Component	Device	CPU	GPU	RAM	OS	Software	TFLOPS
Camera	NVIDIA Jetson TX2	NVIDIA Denver 2 + ARM Cortex-A57	NVIDIA Pascal 256 core	8 GB	Ubuntu 18.04	JetPack 4.6.1	1.3
Edge Server	Dell XPS 8950	Intel i7-12700	NVIDIA GTX 3060	16 GB	Ubuntu 20.04	PyTorch 1.9 CUDA 11.1	12.7
Edge Server	Turing Server	Intel Xeon E5-2620 v4	NVIDIA Tesla P100	64 GB	RHEL 7.9	PyTorch 1.7 CUDA 11.4	21.2

Chapter 3

FastTuner: Fast Resolution and Model Tuning for Multi-Object Tracking in Edge Video Analytics

 This chapter is based on the manuscript: Renjie Xu, Keivan Nalaie, and Rong Zheng. "FastTuner: Fast Resolution and Model Tuning for Multi-Object Tracking in Edge Video Analytics", IEEE Transactions on Mobile Computing, vol. 24, no. 6, pp. 4747–4761, 2025. DOI: 10.1109/TMC.2025.3526573.

3.1 Introduction

Cameras are ubiquitous in our cities nowadays. Vision-based multi-object tracking, a task that aims to estimate trajectories for objects of interest in video feeds, is a pillar of VAPs. It can drive a wide spectrum of downstream applications, such as security surveillance, sports analysis and traffic control.

Conventional MOT methods follow the *tracking-by-detection* paradigm, where object detection and association tasks are performed by separate models [56, 57]. A detection model firstly detects objects (represented as bboxes) in each frame, after which a re-ID model extracts object features (also known as re-ID features) from each bounding box and links the objects based on these features and motion cues. While effective, these two-stage methods have high computation complexity. The separate processing of each bounding box with distinct re-ID models impedes real-time performance when the number of objects is large. With the development of multi-task learning, a recent trend is to combine both tasks in a joint model, where a re-ID branch and a detection branch share the same backbone model [58–61]. By reusing features for both tasks, the inference time is significantly reduced.

However, achieving real-time performance in EVA systems remains a challenge for current MOT methods due to the resource limitations on edge devices and substantial communication overheads in geographically distributed settings. In this study, we aim to accelerate MOT pipelines by tuning configurations. A *configuration* refers to a particular combination of *knobs*. A MOT pipeline can have several knobs, such as frame resolution and backbone model (hereinafter referred to as "model"). The choice of configuration can impact both accuracy and latency. For instance, using a high frame resolution (e.g., 1080p) and a high-complexity model allows accurate detection

of objects but also incurs substantial processing time. The optimal configuration can be defined as the one with the lowest latency whose accuracy is over a desired threshold. The threshold is usually application-dependent and specified by users. Due to the dynamic nature of video content, the optimal configuration varies over time, often at a timescale of minutes or even seconds [10]. For example, we may use a low frame resolution (e.g., 360p instead of 1080p) when objects are close-by and stationary (e.g., at a traffic stop), without impacting the tracking accuracy. In contrast, for distant objects, a higher resolution is needed to maintain accuracy.

Therefore, to achieve a good trade-off between accuracy and latency, one needs to consider the intrinsic dynamics of video feeds and adapt the pipeline configurations accordingly. The key challenge is *how to efficiently identify the optimal configuration at runtime*. The timeliness of such decisions is critical, as any delay may cause them to become outdated, thereby compromising the efficiency of the system. Some existing works [10, 78] employ online profiling to identify the optimal configuration. However, the profiling cost could be prohibitively expensive as the configuration space grows exponentially with the number of knobs and their corresponding values. Despite efforts to accelerate this process by pruning the configuration space, the profiling cost remains significant, hindering efficient configuration selection.

To bridge the gaps, we propose FastTuner, a model-agnostic framework to accelerate MOT pipelines by adapting resolutions and backbone models based on the characteristics of video content. Notably, it differs from the prior works [10, 78] in two aspects. First, FastTuner predicts the performance of different configurations in one shot instead of profiling them one by one. Second, rather than performing the MOT task and the configuration selection task separately, FastTuner unifies the two tasks

by sharing a common backbone model between them. Reusing backbone features for both tasks further improves computation efficiency. This design makes FastTuner model-agnostic, meaning that it can seamlessly work with any fully convolutional network (FCN)-based one-shot tracker (e.g., JDE [59], CenterTrack [60] and FairMOT [61]). In the inference stage, FastTuner runs the most expensive configuration, also known as the *golden* configuration [10] every K frames to determine the optimal configuration based on the heatmaps produced by the detectability branch. Object detection and association are then done by the tracking branch using the selected configuration in the next $K-1$ frames.

In real-world deployments, to facilitate application-specific trade-offs between accuracy and latency, FastTuner affords users tunable parameters, which result in different combinations of resolutions and models. In addition, we design two workload placement schemes between a smart camera and an edge server, that is, Edge Server Only with Adaptive Transmission (SOAT) and Edge Server-Assisted Tracking (SAT). Both schemes take advantage of the reduced frame size and model size offered by FastTuner to decrease the amount of data transmitted over the network and the workloads on the camera.

To evaluate the performance of FastTuner, we conduct experiments on a public MOT dataset and a small-scale testbed consisting of an NVIDIA Jetson TX2 board and a server with Tesla P100 GPUs. FastTuner can achieve a better trade-off between tracking accuracy and latency. In comparison to the state-of-the-art (SOTA) approaches, FastTuner is able to accelerate the computation up to 2.5%–25.5% with a 1.1%–9.2% improvement in MOTA.

In summary, our key contributions are as follows:

- We conduct a quantitative study on the impact of frame resolutions and backbone models on tracking accuracy and latency.
- We propose a model-agnostic framework FastTuner, which aims to accelerate MOT pipelines by adapting frame resolutions and backbone models at runtime.
- We design two workload placement schemes for MOT applications, to accelerate end-to-end processing by taking full advantage of FastTuner's adaptability to reduce the network traffic load and computational load.
- We implement and deploy a prototype of FastTuner on commodity devices for performance evaluation.

3.2 Motivation

In this section, to motivate the design of FastTuner, we study the trade-off between tracking accuracy and latency of MOT pipelines for different frame resolutions and backbone models. Additionally, we investigate how changes in visual content can affect such trade-offs, and discuss the limitations of prior approaches.

3.2.1 Effects of Frame Size and Model Size

In MOT, the selection of input resolution and backbone model can significantly impact the trade-off between tracking accuracy and latency. To understand this, we conduct experiments using a SOTA tracker, FairMOT on a Tesla P100 GPU server using the MOT17 dataset [104]. FairMOT employs DLA-34 [114] as its backbone model,

which is a combination of ResNet-34 and deep layer aggregation (DLA) for multi-scale feature fusion. It also includes two branches for object detection and re-ID. The detection branch, built on CenterNet [115], uses three parallel heads to estimate heatmaps, object center offsets, and bounding box sizes. The re-ID branch estimates re-ID features for each pixel to characterize the object centered at the pixel. The features at the predicted object centers from both branches are used for tracking. By manipulating the number of channels within each layer of DLA-34, we can generate variants of different sizes. Full-DLA-34 represents the original model, while Half- and Quarter-DLA-34 scale this number down by half and a quarter, respectively. Such scaling is widely adopted to generate backbone models of different sizes [43, 116, 117].

Table 3.1 and Table 3.2 show the tracking accuracy (i.e., MOTA) and speed (i.e., Frames Per Second, FPS) of FairMOT with different input resolutions and backbone models. Note that the FPS measurements include data transferring time between CPU and GPU, model inference time, and post-processing time for object association. It is evident that higher resolutions and larger models can produce higher tracking accuracy but require more processing time (and thus lower FPS). However, FPS is not inversely proportional to the frame size and the number of model parameters. Several key factors contribute to this. First, the time to transfer data from CPU to GPU grows non-linearly with data volume, depending on the specific hardware and its optimization. Second, the actual number of multiply-accumulate operations (MACs), represented as FLOPs, is not proportional to the number of parameters in DNNs due to convolution layers. GPU optimization also plays a role. Large volumes of data can exploit the inherent parallel computing capabilities of GPUs, which results in a sub-linear increase in processing time as input data expands. Finally, the post-processing

Table 3.1: MOTA and FPS of FairMOT+Full-DLA-34 with different input resolutions

Resolution (px)	FLOPs	MOTA	FPS
1088×608	72.9B	70.7	16.5
864×480	45.7B	68.8	22.0
704×384	29.8B	65.0	28.1
640×352	24.8B	62.9	28.9
576×320	20.3B	58.9	30.8

Table 3.2: MOTA and FPS of FairMOT with different backbone models at full resolution

Backbone	# Params	FLOPs	MOTA	FPS
Full-DLA-34	20.4M	72.9B	70.7	16.5
Half-DLA-34	5.3M	27.4B	65.5	24.8
Quarter-DLA-34	1.5M	12.0B	60.2	27.9

time is input-dependent and can vary based on the number of detected objects in MOT.

3.2.2 Dynamics of Visual Content

To gain insights on the impact of different configurations on MOT performance over time, we run FairMOT [61] on two video sequences in the MOT17 dataset. To quantify the detection quality, we define *detection rate* as the number of detected objects normalized by the number of ground truth objects. Figure 3.1 illustrates the detection rate of FairMOT with different input resolutions and backbone models. As expected, in both trials, a lower resolution (e.g., 576×320 px) or a smaller model (e.g., Quarter-DLA-34) negatively impacts the detection rate in general. However, for some frames, such as frame 135 and frame 240 in MOT17-09, the detection rates are close for all resolutions and models. This can be explained by the dominance of

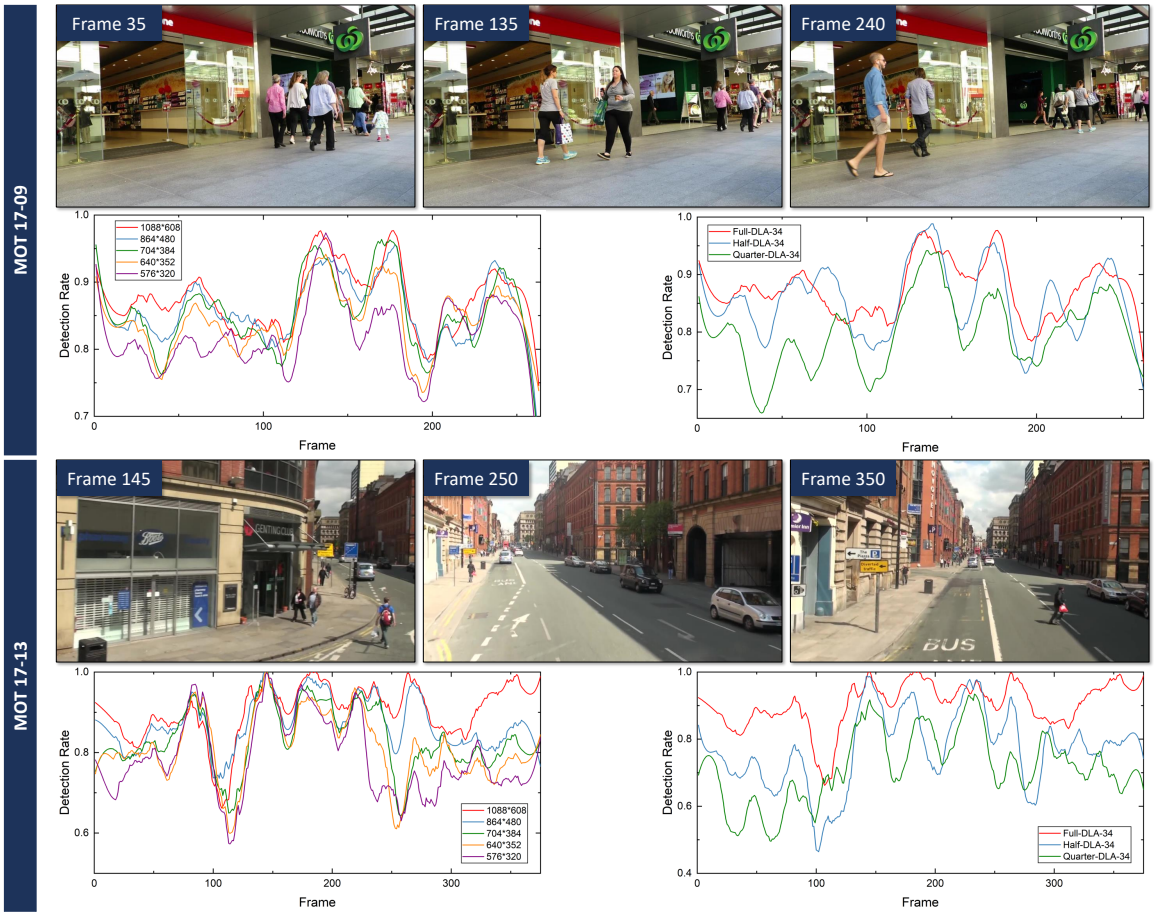


Figure 3.1: Detection rate using FairMOT with different input resolutions and backbone models on two video sequences of MOT17 dataset.

close and bigger objects in those frames. Consequently, lower resolutions and smaller models can be used without compromising accuracy. In contrast, for frame 35 in MOT17-09, since the objects are dense and overlap with each other in the camera view, the detection rates across different configurations vary a lot and only the highest resolution and the largest model can produce good results. Similar observations can be made in MOT17-13. For frame 145, all the configurations can maintain good detection rates, since the objects are sparse and closer. However, for frame 250 and 350 where the objects are far away and small in size, only the most expensive

configuration can yield good results.

In conclusion, these observations reveal that the dynamics in video content offer opportunities for configuration adaptation. It is feasible to accelerate computation by adopting less resource-intensive configurations such as lower resolutions or smaller models, without degrading analytics quality for some scenes.

3.2.3 Efficient and Effective Configuration Decision

Conventional methods [10, 78] rely on separate online profiling to identify the optimal configuration, which is time consuming on resource-limited edge devices. The profiling overhead can grow exponentially with the configuration space. With m knobs and n values per knob, an exhaustive profiling would involve $O(n^m)$ configurations. Various techniques have been proposed to reduce this overhead, such as reducing the search space from $O(n^m)$ to $O(mn)$ by assuming independence among the knobs [10], or further to $O(k)$ by eliminating inferior configurations, where k is much smaller than n^m [15]. However, despite these optimizations, the overhead remains substantial for edge devices, since profiling requires executing the inference pipeline for each candidate configuration. SmartAdapt [15] addresses this challenge by first extracting various features from the image and then training a lightweight DNN to predict the performance of a *given* configuration. However, this approach has its limitations. The selection of features requires domain expertise, and an unsuitable choice may fail to accurately characterize the image content. Additionally, feature extraction for performance prediction for each configuration incurs extra computation time (around 100 ms per run on NVIDIA Jetson TX2, as reported in [15]). Using a dedicated DNN to directly map the image to certain representations that can be used to predict

performance of multiple configurations at the same time can still incur significant overhead. Motivated by the need to drastically reduce online profiling overhead, we propose a novel design that integrates configuration selection and object tracking into a shared model, combining these tasks to achieve both efficiency and accuracy.

Another limitation of most existing works in this line is that they target object detection tasks, where configurations can be switched on a frame-by-frame basis with minimal impact on detection performance. However, this approach is no longer true for MOT tasks, which require consistent object features over time for feature-based object association [59–61]. Frequent and arbitrary changes in resolution or model at the frame level break the consistency of object features across frames, leading to a decline in tracking accuracy. How to effectively perform configuration optimization in MOT tasks remains unexplored.

3.3 Methodology

FastTuner aims to *efficiently* adapt frame resolutions and backbone models based on video content. Switching between models with different architectures (e.g., from ResNet to VGG) [10] poses challenges in MOT tasks, since different model architectures can result in inconsistent re-ID feature distributions, negatively impacting object association [118]. Moreover, the optimal setting for re-ID feature dimension varies across architectures [61], making inter-architecture switching non-trivial as it necessitates feature space alignment [118]. To mitigate these issues, we focus on switching models of different sizes within the same architecture (called model variants), such as switching from a full model to a half model. In real-world applications, it is often more practical to create model variants with different accuracy-latency

trade-offs by pruning the channel dimensions or the number of layers, rather than preparing entirely different models, since the trade-offs of variants tend to be more predictable, whereas entirely different models may produce significantly different and less predictable trade-offs [119].

In contrast to existing works [10, 78] that rely on multiple executions of the pipeline with various configurations, FastTuner reduces the complexity of online profiling through two key ideas. First, we use a single DNN to learn the heatmap representations of different configurations. By integrating the DNN into a tracker, the framework can perform configuration selection and MOT with minimal extra overhead. Second, we use *detection rate* as a surrogate to estimate the tracking performance of different configurations. Tracking performance is intrinsically tied to detection quality—configurations with accurate detection results are likely to yield robust tracking performance [120]. Compared to tracking accuracy, which can only be assessed over multiple frames and with the knowledge of ground truth label, we show in this section that the detection rate of a configuration can be predicted over a single frame using the heatmap associated with the configuration and the golden one.

3.3.1 Algorithm Overview

Figure 3.2 illustrates the FastTuner pipeline. It consists of multiple branches: a detectability branch, a re-ID branch and a detection branch, all sharing a common backbone model. The detection branch contains three heads outputting heatmaps $\hat{H} \in [0,1]^{E\times H'\times W'}$, box sizes $\hat{S} \in \mathbb{R}^{2\times H\times W}$ and center offsets $\hat{O} \in \mathbb{R}^{2\times H\times W}$, where E is the number of object classes ($E = 1$ in this study), $H' = \frac{H}{4}$, $W' = \frac{W}{4}$ and H,

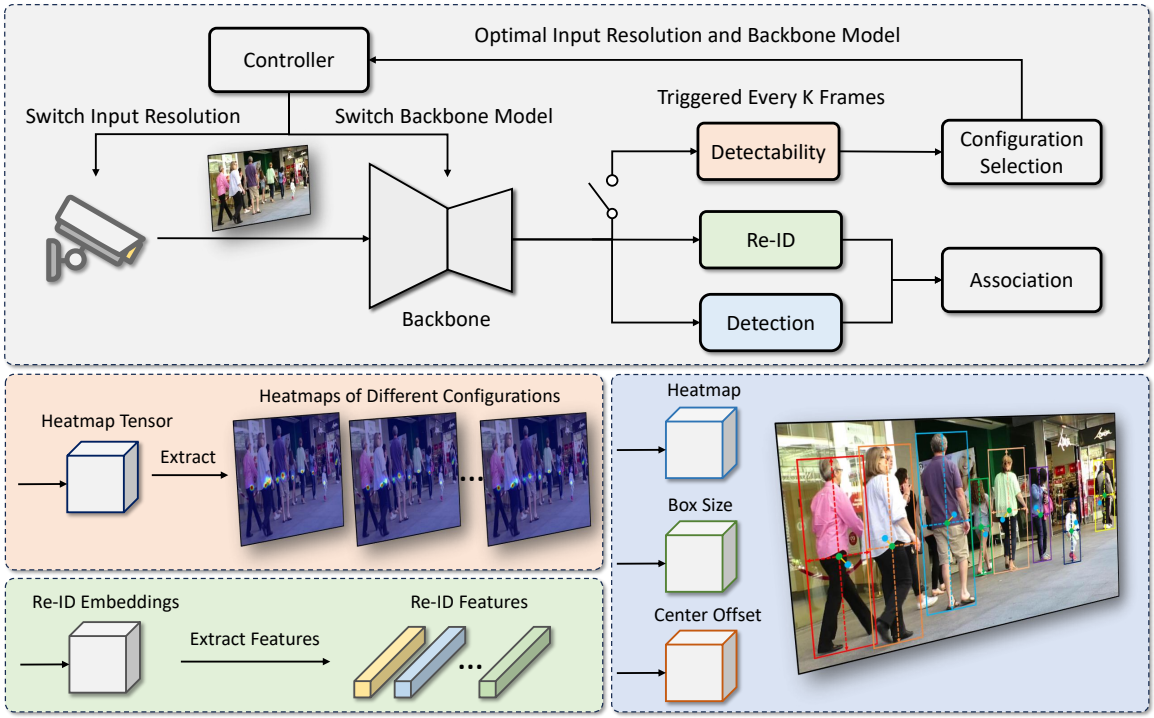


Figure 3.2: Pipeline of FastTuner, with a detectability branch and a tracking branch sharing a common backbone model.

W are the height and width of the input image. A heatmap is a two-dimensional array, where each pixel value in the range of $[0, 1]$ represents the likelihood of an object's center being located at that pixel. The re-ID branch generates re-ID feature embeddings $\hat{E} \in \mathbb{R}^{D \times H \times W}$, where D denotes the feature dimension. The re-ID feature $\hat{E}_{x,y} \in \mathbb{R}^D$ of an object centered at (x, y) can be extracted from the embeddings. The object association is done based on the results from the two branches, following [61]. For simplicity, we use *tracking branch* to refer to the combination of the re-ID and detection branches.

The head in the detectability branch produces the heatmap tensor $\hat{T} \in [0, 1]^{P \times H' \times W'}$, where P denotes the number of configurations. The i-th channel of the tensor is the heatmap $\hat{T}_i \in [0, 1]^{H' \times W'}$ of the configuration $\mathcal{C}_i$, where $i = 1, 2, \ldots, P$. Based on

these heatmaps, the configuration selection module identifies the optimal configuration that can preserve the detectability of the golden configuration (i.e., the most accurate yet computationally expensive configuration) while producing the minimum execution latency.

In the inference stage, to capture the dynamics of video content, the detectability branch is triggered every K frames to identify the optimal configuration, given a full-resolution (FR) input. A controller then switches the input resolution and the backbone model of the pipeline. The subsequent $K-1$ frames are handled by the tracking branch with selected configuration. Notably, any FCN-based one-shot tracker could be adopted in the tracking branch, which makes FastTuner model-agnostic. To demonstrate the feasibility of the idea, we build this branch on top of FairMOT [61], a SOTA tracker. Note that the detectability branch, configuration selection module and controller in Figure 3.2 are the core components of FastTuner, while the remaining are derived from FairMOT.

3.3.2 Multi-Task Learning

As discussed previously, the detectability branch of FastTuner generates heatmap representations of various configurations. This involves learning a transformation $\mathbb{R}^{3\times H\times W} \mapsto [0,1]^{P\times H'\times W'}$, which maps an RGB image to a heatmap tensor. This transformation is learned through several steps. Firstly, a training set comprising images of different resolutions is prepared by resizing the original full-resolution RGB images. Next, a base tracker (e.g., FairMOT) is run on this multi-resolution training set using different backbone models. This generates the centers and corners of the bounding boxes for detected objects in each frame. These coordinates are then

upscaled to match those in a full-resolution frame. Lastly, following the approach of CornerNet [121], a 2D Gaussian filter is applied over each object center. Pixels outside the bounding boxes are penalized according to their distance to the centers of those bounding boxes. Specifically, consider a frame and a detected object within that frame with center coordinates (a_i, b_i) from configuration $\mathcal{C}_i$. Its location on the feature map is obtained by dividing the stride $(a'_i, b'_i) = (\lfloor \frac{a_i}{4} \rfloor, \lfloor \frac{b_i}{4} \rfloor)$. Then the corresponding heatmap response at the location (x_i, y_i) is computed as $Y_{i,x,y} = exp(-\frac{(x_i - a'_i)^2 + (y_i - b'_i)^2}{2\sigma_i^2})$, where a Gaussian kernel is applied and σ_i represents the standard deviation. These heatmaps, constructed offline using the detection results from different configurations, act as ground truth labels to train the detectability branch of FastTuner. In this way, given a full-resolution image, FastTuner can efficiently estimate the heatmap representations of all the configurations in the online stage.

We define the total detectability loss using pixel-wise logistic regression with focal loss [122] as:

$$\mathcal{L}_{\text{detectability}} = -\frac{1}{N} \sum_{i,x,y} \begin{cases} f(\hat{Y}_{i,x,y}) & Y_{i,x,y} = 1; \\ g(Y_{i,x,y}, \hat{Y}_{i,x,y}) & \text{otherwise,} \end{cases} \tag{3.3.1}$$

where

$$\begin{aligned} f(\hat{Y}_{i,x,y}) &= (1 - \hat{Y}_{i,x,y})^{\alpha} \log(\hat{Y}_{i,x,y}), \\ g(Y_{i,x,y}, \hat{Y}_{i,x,y}) &= (1 - Y_{i,x,y})^{\beta} (\hat{Y}_{i,x,y})^{\alpha} \log(1 - \hat{Y}_{i,x,y}), \end{aligned} \tag{3.3.2}$$

$\hat{Y}_{i,x,y}$ is the predicted heatmap of configuration $\mathcal{C}_i$, α and β are pre-set parameters in focal loss (following the setting in [121]), and N is the number of objects in the frame.

Finally, in order to perform configuration selection and object tracking in a shared model, we employ multi-task learning to jointly train the tracking and detectability branches using the following loss function:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{tracking}} + \lambda \mathcal{L}_{\text{detectability}}, \tag{3.3.3}$$

where λ is a pre-determined parameter to balance the two tasks, and $\mathcal{L}_{\text{tracking}}$ is the loss function of the tracking branch defined as:

$$\begin{aligned} &\mathcal{L}_{\text{tracking}} = \\ &\frac{1}{2}\left(\frac{1}{e^{\omega_1}}(\mathcal{L}_{\text{heatmap}} + \mathcal{L}_{\text{box}}) + \frac{1}{e^{\omega_2}}\mathcal{L}_{\text{identity}} + \omega_1 + \omega_2\right), \end{aligned} \tag{3.3.4}$$

which consists of learnable parameters ω_1 and ω_2, heatmap loss, box-size loss, and re-ID loss defined in [61]. This loss function dynamically balances the tasks based on learned uncertainties [123], represented by ω_1 and ω_2. The exponential function ensures these weights remain positive, while the addition of ω_1 and ω_2 acts as regularization, preventing them from increasing too much.

3.3.3 From Heatmaps to Tracking with Adaptive Configurations

In the inference stage, every K frames, the head in the detectability branch predicts heatmaps $\hat{Y}_i$, $i = 1, 2, \ldots, P$. These heatmaps are then converted to their binarized versions $\hat{B}_i \in \{0,1\}^{H' \times W'}$, with pixels over the threshold τ set to 1, otherwise 0. These binarized heatmaps are finally used by the configuration selection module for decision-making.

The problem of selecting the optimal configuration is formulated as:

$$\mathbb{C}' = \left\{ i \in \{1, 2, \ldots, P\} \mid \frac{\sum_{x,y}^{H',W'} \hat{B}_{i,x,y} \cdot \hat{B}_{max,x,y}}{\sum_{x,y}^{H',W'} \hat{B}_{max,x,y}} \geq \gamma_i \right\}, \tag{3.3.5}$$
$$i^* = \text{argmin}_{i \in \mathbb{C}'} L_i,$$

where L_i is the latency of executing configuration $\mathcal{C}_i$, $\hat{B}_{max}$ is the binarized heatmap of the golden configuration, and $\gamma_i \in [0, 1]$ is a user-specified threshold for configuration $\mathcal{C}_i$. γ_i reflects user tolerance of degradation in detectability. If the estimated detection rate of $\mathcal{C}_i$ exceeds the corresponding threshold, this configuration is a candidate configuration. The configuration with the minimum latency in the candidate set $\mathbb{C}'$ will be selected as the optimal configuration $\mathcal{C}_{i^*}$ and applied to the next $K-1$ frames. The latency of each configuration is obtained by offline profiling [10, 9, 7, 78, 16, 79]. Once the optimal configuration is decided, the tracking branch proceeds to extract detections (i.e., bounding boxes) and re-ID features from the heads and perform object association accordingly. For the remaining $K - 1$ frames, only the tracking branch is executed to perform object tracking based on the selected configuration. In Section 3.5.2 we will discuss how to set thresholds to achieve different accuracy-latency trade-offs.

3.3.4 Multi-Resolution Training

We apply multi-resolution training on all the backbone models considered in FastTuner, as it brings several advantages. First, it improves the model's detectability for objects of different sizes caused by either distance or frame sizes. This gain thereby benefits the accuracy of configuration decision making and detection. Second, it can

mitigate the domain gap between different resolutions. Since object tracking consists of object detection and association, tracking performance is affected not only by the detection accuracy, but also informativeness of re-ID features. Changing input resolutions may introduce inconsistencies in re-ID features, which could lead to mismatches in object association and thereby degrade tracking accuracy. However, achieving a balance for all resolutions is challenging, as gains in one resolution can inadvertently compromise the performance of others. To address this challenge, we propose two multi-resolution training schemes:

- **Weighted**: In each epoch, the model is trained on a dataset augmented with all resolutions. Different weights w_r are applied to the loss $\mathcal{L}_r$ prior to model parameter updating $\theta' = \theta - \alpha \times \nabla \mathcal{L}'_r(\theta)$, where $\mathcal{L}'_r = w_r \times \mathcal{L}_r$ and r denotes resolution. Empirically, we set higher weights for higher resolutions. In this way, we guide the model to maintain its proficiency for high resolution inputs, and also pay attention to inputs where it underperforms, typically at lower resolutions.

- **Fine-tune+Weighted**: The model is first trained on the original dataset of the highest resolution, and then fine-tuned on the same dataset but at different resolutions. This fine-tuning process serves to further enhance the model's ability to capture intricate patterns and features during the high-resolution training, and then adapt these learned features to lower resolutions. The combination of weighted loss update ensures that the model's learning is balanced.

.

3.4 Workload Placement on End and Edge Devices

The FastTuner pipeline can be flexibly partitioned between end and edge devices. Compared to conventional computation partition, it introduces a different dimension to balance local computation and network-based processing through configuration adaptation. In this section, two workload partition schemes between a smart camera and an edge server are discussed for edge MOT, as illustrated in Figure 3.3:

- **Edge Server-Only with Adaptive Resolution Transmission (SOAT)**: Every K frames, an FR frame is sent to the server. The server, upon the reception of the frame, performs configuration selection and informs the camera of the appropriate resolution for the subsequent $K-1$ frames to be transmitted. All computations are done on the server, with FastTuner utilized to reduce the transmission time and computational overhead. This setup is beneficial in situations where the camera has sufficient bandwidth, as it requires continuous transmission to the server, ensuring consistent data delivery and processing.

- **Edge Server-Assisted Tracking (SAT)**: An FR frame is sent to the server every K frames. The server processes the frame by computing bounding boxes and re-ID features, and determines the proper resolution for the subsequent frames. These results are then transmitted back to the camera. Upon receiving this information, the camera performs object association. In addition, the camera is responsible for object detection and association for the remaining $K-1$ frames. In this hybrid setup, FastTuner operates partially on the camera and partially on the server, balancing between computational load and bandwidth utilization. This setup is advantageous under limited bandwidth conditions, as

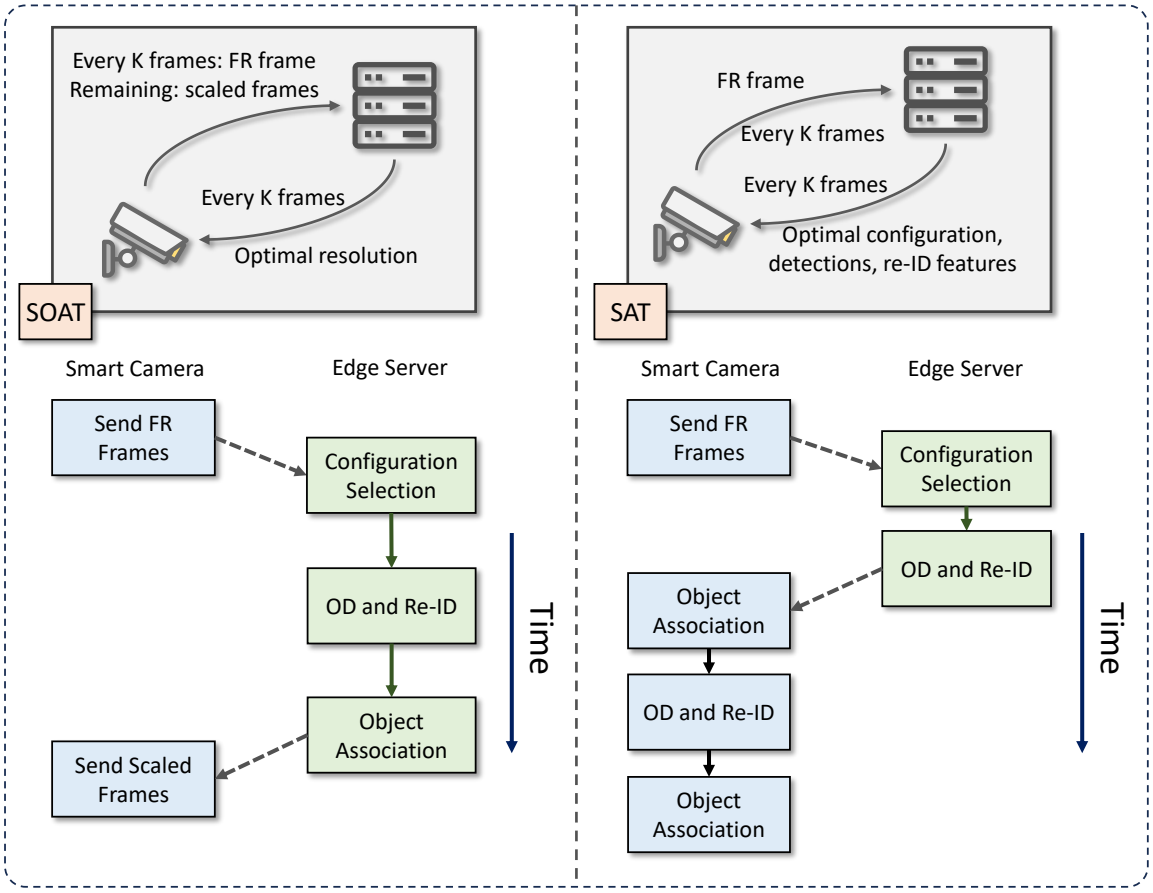


Figure 3.3: Two workload placement schemes, partitioning the workload between a smart camera and an edge server.

it involves the exchange of minimal data in each communication cycle.

Evidently, the two workload placement schemes incur different computation loads on the smart camera and the edge server, and differ in the volume of data transferred over the network. A qualitative summary of these trade-offs is shown in Table 3.3, while a quantitative analysis of experimental results on a real-world testbed is provided in Section 3.5.4.

It is worth noting that in addition to the aforementioned workload placement schemes and control parameter tuning (e.g., threshold and K), further trade-offs

Table 3.3: Qualitative comparison of SOAT and SAT on computation and network loads

Deployments	Computation		Network Traffic	
	Camera	Server	C→S	S→C
SOAT	No	High	Medium	Low
SAT	High	Low	Low	Low

between accuracy and latency can be made by employing different types of backbone models (e.g., ResNet [43], VGG [116]) thanks to the model-agnostic nature of FastTuner. In Section 3.5.2, we additionally evaluate FastTuner using YOLO as its backbone to demonstrate the flexibility.

3.5 Performance Evaluation

In this section, we evaluate the performance of FastTuner on a public MOT dataset and a small-scale real-world testbed.

3.5.1 Implementation

FastTuner is a model-agnostic framework and can easily incorporate any FCN-based object tracking model. In the implementation, we build FastTuner on top of FairMOT [61], by adding a detectability branch consisting of a new head and a configuration selection module. In the inference stage, the tracking branch runs on every frame to perform object detection and tracking while the detectability branch only runs on every K-th frame to select the optimal configuration. The frame resolution and backbone model of FastTuner are configurable. We have five options for resolution: {1088×608, 864×480, 704×384, 640×352, 576×320} px. For the backbone

model, we consider three sizes: {full, half, quarter}. The half and quarter models are generated by pruning the full model, reducing its number of channels in each layer by factors of two and four, respectively. In short, the configuration space of FastTuner comprises $5 \times 3 = 15$ different combinations. To further demonstrate FastTuner's compatibility with different types of backbone models, we also test FastTuner on YOLO, and consider three model sizes as well. Unless otherwise stated, $K = 40$, $\tau = 0.4$, and the results presented in this section are obtained on an NVIDIA Tesla P100 GPU.

Dataset: The MOT17 dataset is used to evaluate the tracking performance. Since the ground truth of test sequences is not public, we follow [60, 109, 118] to split each training sequence into two halves. The first half is used for training while the second half is for validation.

Training: The training of FastTuner is done in two stages. In the first stage, we employ the multi-resolution training methods in Section 3.3.4 to train all the backbone models. Specifically, we employ the *weighted* scheme for training Full-DLA-34 and use the *fine-tune+weighted* scheme for the others. The rationale is that models like Half-DLA-34 and Full-YOLO are less powerful and require good initial weights. In both training schemes, weights of $[1.0, 0.8, 0.6, 0.4, 0.2]$ are applied to the loss function, corresponding to resolutions in descending order, i.e., the highest resolution receives the largest weight, and the lowest receives the least. Then, we follow the steps in Section 3.3.2 to prepare the training data for the detectability branch. In the second stage, we start from the weights in the backbone and the tracking branch of the full model and independently train the detectability branch, while freezing all other parameters. In this way, FastTuner can learn configuration-specific heatmap

representations, without compromising the performance of the tracking branch. We set α, β in Eq. (3.3.2) to 2 and 4, respectively, and set λ in Eq. (3.3.3) to 1.

Metrics: We use MOTA to measure the tracking accuracy of MOT methods, and FPS to measure their runtime speed, indicating the number of frames processed per second. There is often a trade-off between MOTA (accuracy) and FPS (speed)—increasing one may decrease the other. A better tracker should produce a better trade-off.

3.5.2 Comparison with Baselines

Baselines: For comparison, we implement FairMOT, equipped with different backbone architectures (DLA-34 and YOLO) of different model sizes (full, half, quarter). They are called FairMOT+{Full, Half, Quarter}-DLA-34 and FairMOT+{Full, Half, Quarter}-YOLO.

Adaptive vs. Fixed Configuration: In Figure 3.4, we show the tracking performance (MOTA) and speed (FPS) of FastTuner and the baselines on the MOT17 dataset, with results obtained on both an NVIDIA Tesla P100 (a) and a GTX 1060 (b). The baselines, i.e., FairMOT with Full-, Half- and Quarter-DLA-34 are presented as distinct curves using quadratic fitting. The data points on these curves depict the MOTA-FPS pairs achieved at different resolution levels, ordered from high to low as: 1088×608, 864×480, 704×384, 640×352, 576×320. Each baseline encompasses a range of trade-offs between MOTA and FPS. Notably, on both devices, FastTuner consistently outperforms the baselines, with its curve positioned to the upper right of all others, indicating a superior trade-off between MOTA and FPS. For a given FPS, FastTuner achieves a higher MOTA, while for a specified MOTA, it delivers

a higher FPS. The points in FastTuner are obtained by different threshold settings in Table 3.4, where the "-" symbol means that the corresponding configuration is excluded for selection. Restricting candidate configurations helps avoid ones that overly degrade tracking performance and reduce frequent switching, which may result in many inconsistent ID features.

The comparisons clearly demonstrate the advantage of configuration adaptation using FastTuner. For instance, in Figure 3.4a, FastTuner+T1, while maintaining a similar MOTA as FairMOT+1088 × 608 px+Full-DLA-34, is 10.3% faster. Furthermore, FastTuner+T4 sees a 3.8% increase in MOTA and a 5.9% increase in FPS by adapting both resolutions and models, compared to FairMOT+640 × 352 px+Full-DLA-34. Overall, FastTuner achieves 1.0%–4.2% improvements in MOTA and 0.7%–10.3% in FPS, compared to the baselines. Similar improvements, i.e., 1.0%–7.0% in MOTA and 1.6%–14.5% in FPS, can be observed in Figure 3.4b. The choice of the threshold setting should be based on application requirements. For example, if a higher tracking accuracy is prioritized over latency, FastTuner+T1 could be the best option. On the contrary, if speed is more important, FastTuner+T7 would be more suitable.

To understand how gains in MOTA or FPS or sometimes both are achieved, we further provide a breakdown of the percentage of the configurations selected by FastTuner under different threshold settings in Figure 3.7. For example, FastTuner+T1 processes 23.5% of frames at the lower resolution of 864 × 480 px, a noticeable contrast to the baseline that processes all frames at 1088 × 608 px. Similarly, around one quarter of the total frames are processed by ligher configurations in FastTuner+T4, such as 864 × 480 px+Half-DLA-34 and 576 × 320 px+Full-DLA-34. This trend is

consistent across the other threshold settings.

Impact of Interval K: Recall that the detectability branch of FastTuner is triggered every K frames, and the selected configuration is applied to the next $K-1$ frames. Next, we investigate how different values of K can affect the tracking accuracy and speed of FastTuner. The impact of K on the MOTA and FPS of FastTuner under different threshold settings is shown in Table 3.6. It is evident that FPS decreases when K decreases as the golden configuration (i.e., 1088×608 px+Full-DLA-34) is executed more frequently. Somewhat counter-intuitively, performing configuration selection more frequently (i.e., when K is smaller) is not guaranteed to improve MOTA. For instance, in FastTuner+T6, when K decreases from 40 to 10, the MOTA slightly decreases from 55.1% to 54.7%. This could be explained by the inconsistencies of re-ID features caused by frequent configuration changes. Therefore, one should be careful in setting K, since an optimal K depends on many factors, such as threshold settings, video content characteristics, application requirements, etc.

Compatibility with Different Backbone Networks: In Figure 3.5, we utilize YOLO as the backbone model in FastTuner. As shown, the resulting trends are quite similar to those observed previously with DLA-34 in Figure 3.4. However, as YOLO is much more efficient yet less powerful than DLA-34, there is a distinct shift in the MOTA–FPS range, with notably higher FPS but lower MOTA. Overall, FastTuner (YOLO) achieves 1.0%–7.0% improvements in MOTA and 0.2%–3.6% in FPS on Tesla P100, and 1.5%–6.5% higher FPS on GTX 1060, compared to the baselines. The pre-defined configuration settings for this architecture are summarized in Table 3.5. The percentage of different configurations in FastTuner (YOLO) are presented in Figure 3.8. It is worth noting that in certain settings, such as T5 and T6, the

slices are quite narrow. This is primarily because we prioritize tracking accuracy by tightening the threshold settings. One can relax the settings to achieve a higher FPS, but at the cost of tracking accuracy. Lastly, the impact of K is examined in Table 3.7, and the observations align with those from Table 3.6.

3.5.3 Comparison with SOTA approaches

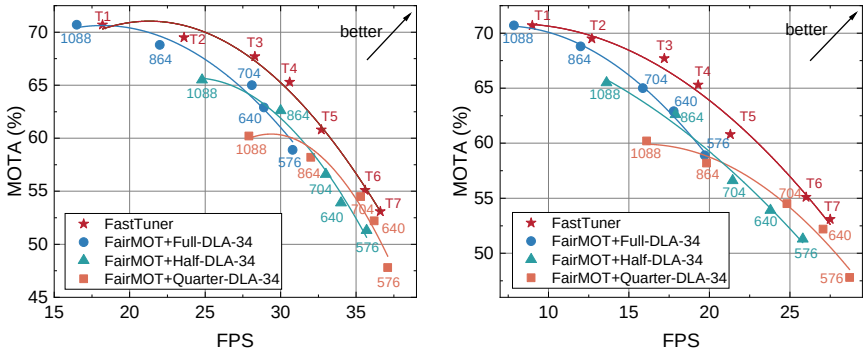
In this section, we compare FastTuner with three SOTA approaches. To ensure a fair comparison, we adapt these methods, originally designed for detection tasks, into trackers. The tracking components and configuration space are made consistent with FastTuner, but configuration decisions are based on the original design.

- **VideoStorm** [9]: VideoStorm exhaustively profiles all configurations on the first K frames of a video, selects the cheapest configuration that meets the accuracy requirement (i.e., accuracy $\geq \alpha$), and uses this configuration for the remaining video. The accuracy requirement α can be tuned to achieve different accuracy-latency trade-offs. In our implementation, we set $K = 40$.

- **Chameleon** [10]: Chameleon conducts exhaustive online profiling every T frames to identify top-k best configurations (i.e., the k cheapest configurations with accuracy $\geq \alpha$). Then, for every K frames, it selects the optimal configuration from this subset through profiling (called *partial profiling*). The results from the golden configuration are used as ground truth to measure accuracy. Additionally, Chameleon assumes that knobs contribute independently to the accuracy in order to significantly reduce the exploration space. In our experiments, we use $T = 120$, $K = 40$, $k = 3$.

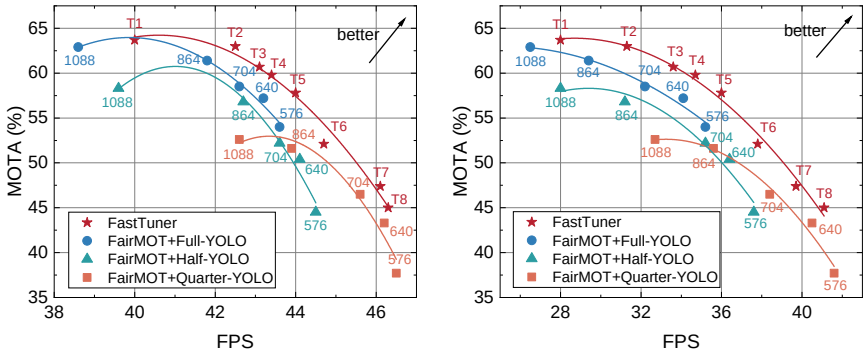
- **SmartAdapt** [15]: SmartAdapt utilizes a content feature extractor to establish a mapping $f(\cdot)$ from a frame X to its feature representation $f(X)$. It then applies a content-aware accuracy prediction model to build a mapping $a(\cdot)$ from the feature representation $f(X)$ to the accuracy of a given configuration c, expressed as $a(c, f(X))$. The cheapest configuration that meets the accuracy requirement will be selected. The data used to train the accuracy predictor is obtained through offline profiling. The optimal configuration is decided every K frames, where $K = 40$. For $f(\cdot)$, we use lightweight features (average object width, height and number), feature embeddings extracted from HoC, HOG algorithms, the backbone (i.e., DLA-34), and a separate feature extractor (i.e., YOLOv8n).

Figure 3.6 shows the MOTA and FPS of FastTuner and the SOTA methods on the MOT17 dataset, with results obtained on both a Tesla P100 (a) and a GTX 1060 (b). α_1 to α_5 represent different accuracy requirement settings, with values of 0.9, 0.8, 0.7, 0.6, and 0.5, respectively. Larger α's prioritize accuracy while smaller ones sacrifice accuracy for speed. Among these methods, FastTuner achieves the best MOTA-FPS trade-offs. Due to the non-adaptive strategy of VideoStorm, it fails to capture the dynamics in the scenes as the optimal configuration can vary over time. The best configuration selected in the beginning may become ineffective in later frames. Interestingly, Chameleon performs worse than VideoStorm, despite its adaptive mechanism. This could be attributed to three factors. First, the profiling cost is significant, as it requires periodic online profiling. Exhaustive profiling involves executing the pipeline 15 times, while partial profiling requires 3 executions. Second, the selection



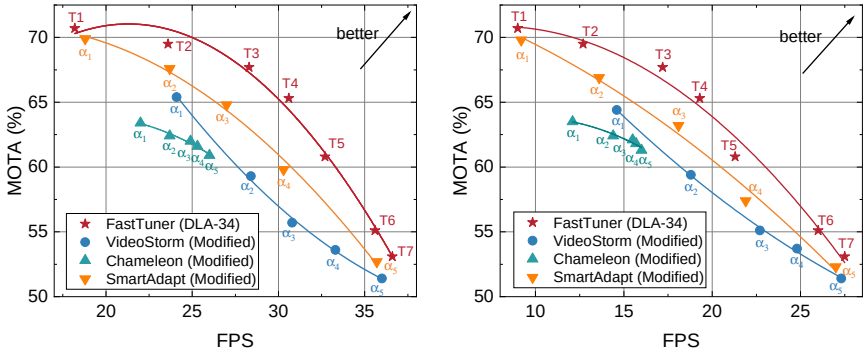
(a) Tesla P100 (b) GTX 1060

Figure 3.4: Comparison between FastTuner (DLA-34) and the baselines: FairMOT+{Full, Half, Quarter}-DLA-34 on MOT17 across two devices.



(a) Tesla P100 (b) GTX 1060

Figure 3.5: Comparison between FastTuner (YOLO) and the baselines: FairMOT+{Full, Half, Quarter}-YOLO on MOT17 across two devices.



(a) Tesla P100 (b) GTX 1060

Figure 3.6: Comparison between FastTuner (DLA-34) and three SOTA approaches: VideoStorm, Chameleon and SmartAdapt on MOT17 across two devices.

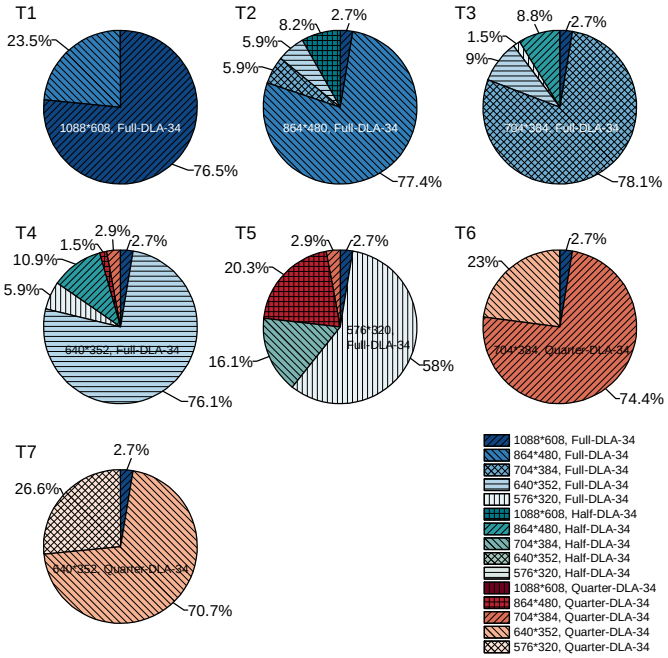


Figure 3.7: Percentages of the configurations selected by FastTuner (DLA-34) under different threshold settings: T1–T7.

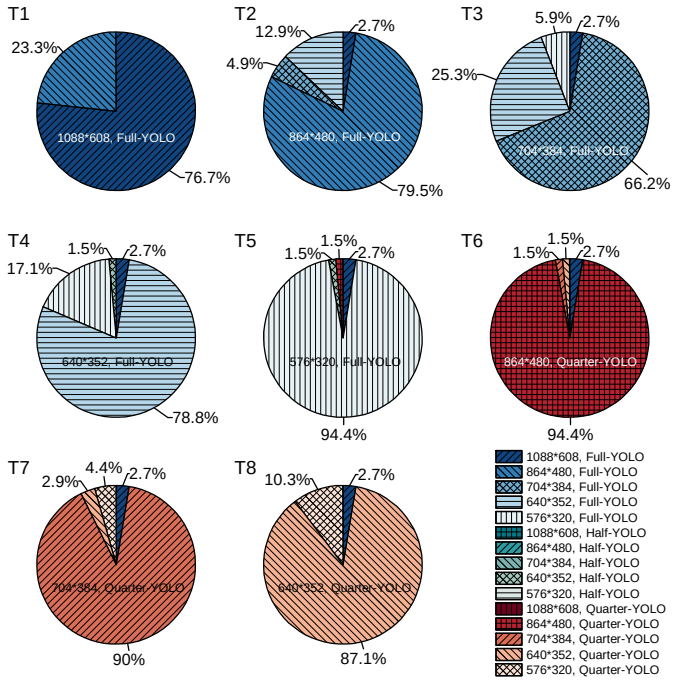


Figure 3.8: Percentages of the configurations selected by FastTuner (YOLO) under different threshold settings: T1–T8.

Table 3.4: Threshold settings in FastTuner (DLA-34) and corresponding results

Threshold	Full-DLA-34					Half-DLA-34					Quarter-DLA-34					MOTA	FPS
	1088	864	704	640	576	1088	864	704	640	576	1088	864	704	640	576		
T1	-	0.99	-	-	-	-	-	-	-	-	-	-	-	-	-	70.7	18.2
T2	-	0	0.95	0.99	-	0.99	-	-	-	-	-	-	-	-	-	69.5	23.6
T3	-	-	0	0.85	0.95	-	0.95	-	-	-	-	-	-	-	-	67.7	28.3
T4	-	-	-	0	0.85	-	0.9	0.9	-	-	-	0.9	0.9	-	-	65.3	30.6
T5	-	-	-	-	0	-	-	0.8	0.9	-	-	0.8	0.9	-	-	60.8	32.7
T6	-	-	-	-	-	-	-	-	-	-	-	-	0	0.65	-	55.1	35.6
T7	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0.6	53.1	36.6

Table 3.5: Threshold settings in FastTuner (YOLO) and corresponding results

Threshold	Full-YOLO					Half-YOLO					Quarter-YOLO					MOTA	FPS
	1088	864	704	640	576	1088	864	704	640	576	1088	864	704	640	576		
T1	-	0.99	-	-	-	-	-	-	-	-	-	-	-	-	-	63.7	40.0
T2	-	0	0.95	0.99	-	-	-	-	-	-	-	-	-	-	-	63.0	42.5
T3	-	-	0	0.85	0.95	-	-	-	-	-	-	-	-	-	-	60.7	43.1
T4	-	-	-	0	0.85	-	-	-	0.9	-	-	-	-	-	-	59.8	43.4
T5	-	-	-	-	0	-	-	-	0.85	-	-	0.85	-	-	-	57.8	44.0
T6	-	-	-	-	-	-	-	-	-	-	-	0	0.7	0.7	-	52.1	44.7
T7	-	-	-	-	-	-	-	-	-	-	-	-	0	0.65	0.65	47.4	46.1
T8	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0.5	45.0	46.3

Table 3.6: Impact of interval K on FastTuner (DLA-34)

Threshold	K=40		K=20		K=10		K=2	
	MOTA	FPS	MOTA	FPS	MOTA	FPS	MOTA	FPS
T1	70.7	18.2	70.8	17.8	70.9	17.6	70.9	17.0
T2	69.5	23.6	69.5	23.3	69.6	22.8	70.8	19.2
T3	67.7	28.3	67.2	28.1	67.3	27.2	69.5	20.7
T4	65.3	30.6	65.1	30.0	65.1	28.5	68.3	21.0
T5	60.8	32.7	60.2	31.4	60.0	30.0	67.0	21.3
T6	55.1	35.6	54.9	33.9	54.7	32.1	61.5	21.4
T7	53.1	36.6	53.0	34.5	53.0	32.7	60.6	21.5

Table 3.7: Impact of interval K on FastTuner (YOLO)

Threshold	K=40		K=20		K=10		K=2	
	MOTA	FPS	MOTA	FPS	MOTA	FPS	MOTA	FPS
T1	63.7	40.0	63.7	39.6	63.6	39.3	63.5	38.7
T2	63.0	42.5	63.2	41.9	63.2	41.5	63.7	40.4
T3	60.7	43.1	60.7	42.6	60.7	42.2	62.1	40.9
T4	59.8	43.4	59.9	43.2	59.8	42.9	61.7	41.2
T5	57.8	44.0	57.7	43.5	58.0	43.2	60.3	41.5
T6	52.1	44.7	51.9	44.3	51.7	44.1	55.9	41.7
T7	47.4	46.1	47.3	45.7	47.2	45.0	53.0	41.9
T8	45.0	46.3	44.6	46.0	44.6	45.8	51.8	42.1

of top-k configurations could be outdated, or inaccurate (especially when the independence assumption does not hold), leading to inferior configurations. Third, frequent changes in the top-k configurations result in varying selections, which can disrupt the consistency of object features during object association. SmartAdapt removes inferior configurations identified through offline profiling from online selection. Doing so allows it to achieve better MOTA compared to Chameleon. However, it still suffers from substantial overhead due to feature extraction, as it involves running HoC, HOG algorithms, and a DNN feature extractor. In contrast, FastTuner incurs only a minimal overhead from running the detectability branch, which is much more efficient. In summary, FastTuner outperforms SmartAdapt on both devices, with 1.1%–9.2% higher MOTA and 2.5%–25.5% higher FPS on Tesla P100, and 1.3%–7.1% higher MOTA and 1.9%–38.4% higher FPS on GTX 1060.

3.5.4 Testbed Evaluation

Setup: Our testbed consists of the NVIDIA Jetson TX2, the Turing server, and the Dell desktop, as introduced in Section 2.5. The Jetson TX2 serves as the smart camera, while the server or desktop serves as the edge server. We evaluate the system performance under different network conditions by enabling camera-server communication via Ethernet, unrestricted Wi-Fi (Wi-Fi-H), and Wi-Fi with a 5 Mbps limit (Wi-Fi-L) to emulate different connectivities [124, 28, 125, 126]. The average upload and download bandwidths and round-trip time (RTT) of the networks are given in Table 3.8.

Implementation: Due to the resource constraints of the NVIDIA Jetson TX2, we only consider FastTuner (YOLO) on the testbed. For simplicity, for all the network

Table 3.8: Metrics of the networks

Network	Upload (Mbps) (camera→server)	Download (Mbps) (server→camera)	RTT (ms)
Ethernet	769.0	938.0	1.1
Wi-Fi-H	21.4	41.5	15.8
Wi-Fi-L†	4.5	4.6	10.6

† The bandwidths are constrained to a maximum of 5 Mbps.

settings, we evaluate FastTuner under three threshold settings: T1, T4 and T8. The communication between the camera and the server uses Transmission Control Protocol (TCP). To reduce the network traffic load, tools from the OpenCV library [40] are used to compress the frames before sending them over the network. Upon reception of the frames, the server decompresses them before further processing.

Baselines: We consider two baselines for comparison: Baseline-Camera (B-C) with all computations done locally on the camera, and Baseline-Server (B-S) with each frame transmitted to the server for computation. Note that both baselines do not incorporate FastTuner. We also consider an additional scheme called Camera-Only (CO), which incorporates FastTuner but places all workloads on the camera. To ensure a fair comparison with FastTuner+T1, T4 and T8, the baselines employ the configurations of FairMOT+Full-YOLO+1088 × 608 px (C1), FairMOT+Full-YOLO+640 × 352 px (C2), and FairMOT+Quarter-YOLO+640 × 352 px (C3), respectively.

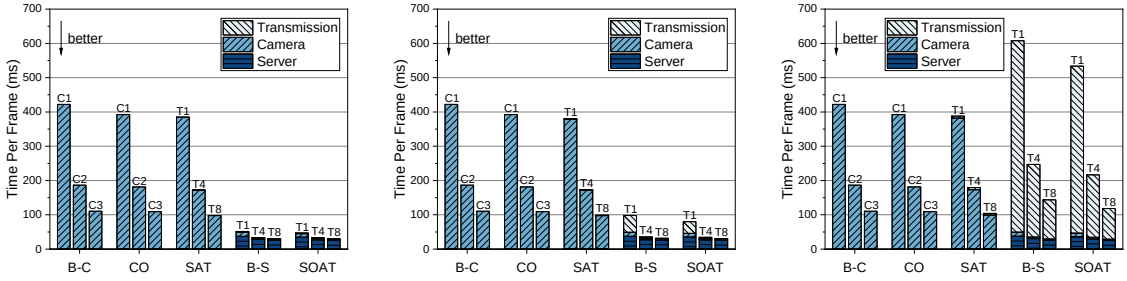
Comparisons among workload placement schemes: Figure 3.9 provides a breakdown of the time spent on each part (i.e., server, camera and transmission) across the five schemes on the testbed. The camera time and server time represent the computational time spent on the camera and the server, respectively. The

transmission time includes the upload time (camera→server) and download time (server→camera). In all experiments, K is set to 40.

Under high bandwidth conditions (i.e., Ethernet and Wi-Fi-H), as indicated in Figure 3.9a and Figure 3.9b, CO surpasses B-C by taking advantage of the configuration selection in FastTuner to reduce computational time on the camera. SAT enhances this advantage since the server handles the most time-consuming task, namely object detection [109]. Given sufficient network bandwidths, B-S is much faster than camera-centric deployments (i.e., B-C, CO, and SAT) by transmitting compressed full-resolution frames with minimal overhead. With FastTuner, SOAT further accelerates the overall processing by transmitting lower-resolution frames and utilizing smaller models for inference. Consequently, SOAT is the optimal scheme under Ethernet, delivering an FPS in the range of 21.2 to 33.4 and marking a 1.7%–8.7% acceleration in comparison with B-S. The gap widens to 4.1%–22.5% under Wi-Fi-H, as the transmission time savings brought by FastTuner are more significant.

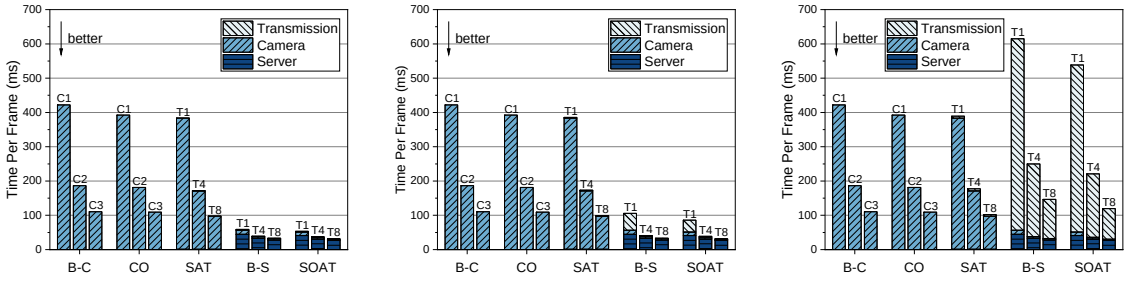
In contrast, when the available bandwidth is limited (i.e., using Wi-Fi-L), as depicted in Figure 3.9c, server-centric schemes (i.e., B-S and SOAT) experience significant slowdowns since network communication becomes a bottleneck. SAT, however, is less affected and still manages to outperform B-C and CO since only lightweight information, such as bounding box coordinates (i.e., detections) and vectors (i.e., re-ID features) is communicated. Therefore, SAT proves to be the best scheme, achieving 2.6–9.6 FPS and 3.7%–8.3% speed-up over B-C.

Additionally, the observations derived from GTX 1060, as shown in Figure 3.10, are consistent with those in Figure 3.9. In short, when the network bandwidth is sufficient, i.e., using Ethernet and Wi-Fi-H, SOAT has the best performance, with



(a) Ethernet (b) Wi-Fi-H (c) Wi-Fi-L

Figure 3.9: Comparisons between different schemes on the testbed (Tesla P100) across three different networks.



(a) Ethernet (b) Wi-Fi-H (c) Wi-Fi-L

Figure 3.10: Comparisons between different schemes on the testbed (GTX 1060) across three different networks.

an improvement of 5.3%–22.1% in speed. However, when the bandwidth is limited, i.e., using Wi-Fi-L, SAT emerges as the best scheme, accelerating the pipeline by 3.7%–8.3%. These two experiments demonstrate that FastTuner can accelerate end-to-end processing across networks with varying bandwidths and devices with different computing power.

3.6 Conclusion

In this chapter, we proposed FastTuner, a model-agnostic framework to accelerate MOT pipelines in EVA systems. By learning heatmap representations of different

configurations offline, FastTuner can intelligently select the optimal frame resolution and backbone model at runtime, improving the trade-off between tracking accuracy and speed. The integration of multi-task learning allows configuration selection and object tracking to be performed in a shared model, further reducing the computational overhead. FastTuner offers users opportunities to control the trade-off based on the application requirements by tuning its threshold settings. For real-world deployments, we designed two workload placement schemes between a smart camera and an edge server. Extensive experiments demonstrate that 1) FastTuner can work with different backbone models; 2) it can improve the MOTA-FPS trade-off of MOT pipelines in real-world EVA systems.

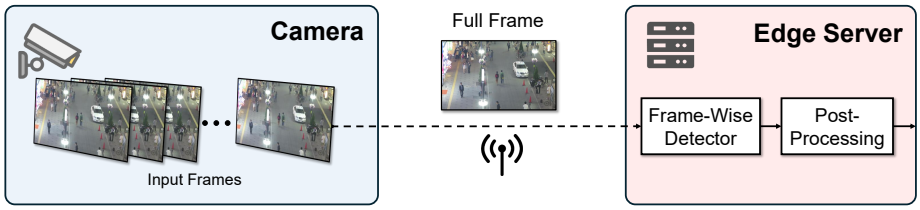
Chapter 4

BlockHybrid: Accelerating Object Detection Pipelines with Hybrid Block-Wise Execution

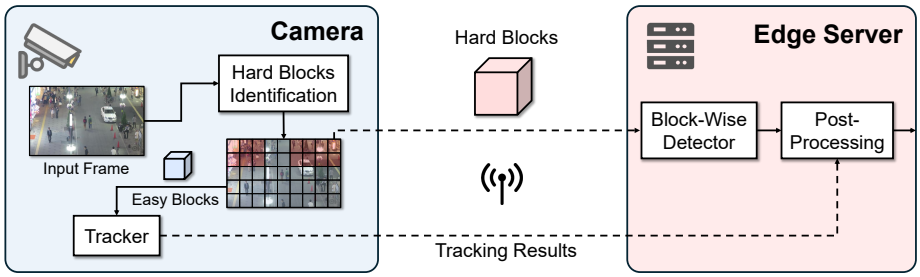
4.1 Introduction

The proliferation of IoT cameras is changing our lives. According to [3], nearly 42 billion IoT devices will generate 79.4 ZB of data by 2025, 80% of which will be video or video-like. While such vast data unlock valuable insights for video analytics applications, they also pose significant challenges, especially in latency-sensitive scenarios. For instance, in ITS [33], roadside units (RSUs) play a critical role in monitoring pedestrians and vehicles, providing real-time warnings to connected vehicles via vehicle-to-everything (V2X) communication [33]. To prevent accidents and ensure smooth traffic flow, RSUs must process high-FPS videos streamed from traffic cameras with low latency—even slight delays in detecting jaywalkers or sudden lane changes can be dangerous, while slow congestion updates may impair traffic efficiency. These scenarios highlight the need for efficient, real-time edge video analytics to enhance road safety and traffic management.

Object detection serves as the foundation of many VAPs, with its outputs directly driving downstream tasks such as object tracking and event analysis. This critical role has made its optimization, particularly for edge deployment, an active research topic. A conventional object detection pipeline is illustrated in Figure 4.1a. A camera offloads every captured frame to an edge server hosting a frame-wise object detector for processing. This pipeline is well-suited for latency-sensitive applications such as ITS, where rapid response is critical. By leveraging frame-wise streaming, it enables real-time capturing, encoding, transmission, and processing of each frame, ensuring timely detection of pedestrians, vehicles, and obstacles for instant alerts and traffic control. Many methods have been proposed to optimize this pipeline, including model compression [127, 128] and specialization [129, 130] to reduce computational complexity,



(a) Conventional pipeline



(b) Proposed pipeline

Figure 4.1: Comparison between conventional pipeline and the proposed pipeline

.

model partitioning [131–135, 63, 136–144] to distribute inference workloads between the camera and the edge server, and model cascade techniques [145–150, 4, 151, 118] that progressively refine predictions through multiple stages to balance accuracy and efficiency.

However, the aforementioned methods are not input-aware. Videos often contain substantial redundancy, and not every pixel is worth transmitting and processing. Imagine a traffic monitoring application deployed at a busy intersection: non-informative regions, such as empty roads, offer little useful information and require minimal computation, whereas regions of interest, with vehicles or pedestrians, demand accurate processing. This contrast highlights the need for selective processing strategies, where communication and computation resources are allocated according to the significance of different frame regions—a technique known as *conditional execution*. When the execution decisions are made on a block-by-block basis, this

technique is called *block-wise conditional execution*. Relevant approaches [19, 152] divide input frames into blocks of equal size, identify the informative ones (i.e., those containing objects of interest), and process them using a customized block-wise detector, where the features of informative blocks are computed, and uninformative blocks reuse features from the previous execution before merging to generate the final results. However, a key limitation is that they treat all informative blocks equally, ignoring differences within the blocks. For example, a block containing only a slow-moving pedestrian could be efficiently handled by a simpler algorithm, rather than a heavy detector. This one-size-fits-all strategy leads to unnecessary resource consumption, leaving room for further optimization. Another challenge of block-wise detection is the potential for *block artifacts* [20], which arise when features from new and cached blocks are merged during inference. The inconsistencies at block boundaries can result in inaccurate or redundant detections, particularly in regions where objects cross block boundaries.

To address these challenges, we propose BlockHybrid, a novel framework for accelerating object detection pipelines using hybrid block-wise conditional execution. In this framework, blocks are further categorized into *hard* and *easy* blocks based on their content and processing strategy. Easy blocks include informative blocks with easy objects (e.g., a slow-moving pedestrian) or entirely non-informative blocks (e.g., empty roads) that can be accurately processed using lightweight algorithms. In contrast, hard blocks are a subset of informative blocks containing challenging objects that require heavy computation. A brief pipeline of BlockHybrid is depicted in Figure 4.1b. BlockHybrid begins by dividing each frame into multiple uniform blocks and labels them as hard or easy blocks with a policy network. Hard blocks are transmitted

to the server and processed by a block-wise detector, while easy blocks are handled locally on the camera using a lightweight tracker. By selectively transmitting and processing only hard blocks, redundant computation and communication are significantly reduced. We define this hybrid block-wise execution strategy as *block-hybrid* conditional execution. To further mitigate block artifacts, we introduce block-wise fine-tuning, an additional training stage applied to the detector. This stage simulates scenarios with object boundaries across blocks and improves the model's ability to handle feature inconsistencies.

To evaluate the performance of BlockHybrid, we conduct experiments on two public object detection benchmarks. We also implement and deploy a prototype on a real-world testbed that uses an NVIDIA Jetson TX2 as a camera and an Ubuntu desktop as an edge server running object detection tasks. Our results show that BlockHybrid achieves a better trade-off between detection accuracy and end-to-end latency. Compared to SOTA approaches, BlockHybrid improves execution speed by 8.8%–31.5% while maintaining comparable accuracy.

In summary, our key contributions are as follows:

- We conduct a quantitative analysis of video redundancy and explore the relationship between the number of hard blocks and execution latency.
- We propose a novel framework, BlockHybrid, which accelerates object detection pipelines through block-hybrid conditional execution.
- We design a policy network, trained offline based on reinforcement learning, to make online block decisions.

- We introduce block-wise fine-tuning to address block artifacts and improve detection accuracy during block-wise execution.
- We implement and deploy a working prototype of BlockHybrid on a real-world testbed, demonstrating its performance and effectiveness.

4.2 Motivation

In this section, we conduct two experiments to show the redundancy in videos and the possible improvements made by block-hybrid execution.

4.2.1 Redundancy in Videos

Videos often contain significant redundancy. Hard blocks that require full-fledge detectors only occupy a small proportion of the frame, with the rest dominated by easy blocks (e.g., background, sparse objects, etc.). We measure the redundancy, quantified as the number of informative blocks (IBs) and hard blocks (HBs), of two datasets: MOT17 [104] and WildTrack [105] in Table 4.1. Each frame is resized to 1024×2048 and divided into 8×16 uniform blocks (i.e., 128×128). In every $T = 10$ frames, we mark the first frame as the key frame and the following $T - 1$ frames as regular frames. We use ground truth labels to obtain the bboxes in each frame, and those in key frames are referred to as *references*. We then use a tracker (e.g., Median Flow [153]) to track the references in the subsequent frames. The tracking results are compared with the ground truth using IoU. The bboxes with an IoU below a specified threshold (e.g., 80%) are considered hard objects, since they are hard to be tracked accurately. The corresponding blocks are defined as hard blocks. A block is identified

Table 4.1: Redundancy of MOT17 and WildTrack datasets.

Sequence	**# Frame**	**Density**[1]	**IBs**[2]	**HBs**[2]
MOT17-02	600	30.97	28.37%	13.39%
MOT17-04	1050	45.29	50.96%	8.28%
MOT17-05	837	8.26	78.31%	31.82%
MOT17-09	525	10.14	36.90%	18.65%
MOT17-10	654	19.63	26.93%	14.29%
MOT17-11	900	10.48	50.95%	12.74%
MOT17-13	750	15.52	13.85%	7.64%
WildTrack-Cam1	2000	35.76	37.42%	17.40%
WildTrack-Cam2	2000	24.89	50.63%	19.02%
WildTrack-Cam3	2000	38.27	62.38%	33.61%
WildTrack-Cam4	2000	29.30	34.47%	15.02%
WildTrack-Cam5	2000	27.04	41.38%	16.56%
WildTrack-Cam6	2000	34.19	35.91%	14.63%
WildTrack-Cam7	2000	30.85	40.63%	13.26%

[1] Density denotes the average number of objects per frame.

[2] IB and HB refer to informative block and hard block, respectively.

as an informative block if it contains any bbox.

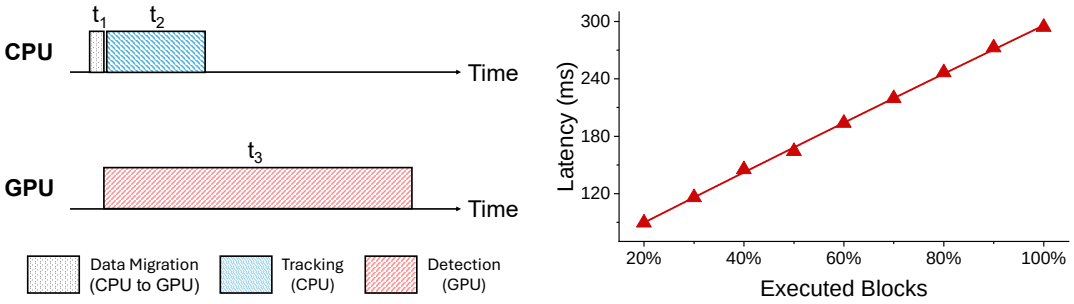
In Table 4.1, it is evident that informative blocks typically constitute no more than 50% of video frames, and in some sequences, e.g., MOT17-05, their proportion drops below 10%. However, hard blocks account for an even smaller percentage, generally less than 20%, highlighting the substantial redundancy present in videos. By focusing only on transmitting and processing hard blocks, communication and computation overhead can be significantly reduced. This observation motivates us to design an efficient method to judiciously identify and handle hard blocks while maintaining accuracy.

4.2.2 Potential Acceleration with Hybrid Block-Wise Execution

Does reducing the number of detection blocks proportionally translate to computational savings? To answer this question, we follow [19] to perform block-wise detection, with CSP [154] as the detector and ResNet-50 as its backbone. Median Flow [153] is used as the tracker to handle objects within easy blocks. Considering the efficiency of Median Flow on CPUs, we carefully schedule these two workloads as depicted in Figure 4.2a. The hard blocks are first migrated from the CPU to the GPU. Once the GPU begins processing the hard blocks, the CPU simultaneously runs the tracker. This parallel execution results in only negligible additional overhead.

We measure the latency from the start of data migration to the completion of tracking and detection for each frame, represented as $t_1 + \max(t_2, t_3)$ in Figure 4.2. Clearly, the latency is almost proportional to the number of hard blocks. In particular, executing 20% of the blocks leads to a reduction of 205 ms in latency, only 30% of the original latency. This observation shows the potential for accelerating object detection pipelines by only executing a small subset of blocks.

However, achieving effective block-hybrid execution is non-trivial. First, differentiating between hard and easy blocks using a policy network is challenging and requires a carefully designed reward function. Second, the decision process must remain efficient; otherwise, its overhead could negate the benefits of reducing the number of executed blocks. Finally, performing accurate block-wise detection is difficult due to block artifacts, which can degrade the overall detection performance. An example of block artifacts is shown in Figure 4.3. Hard blocks are highlighted in red, while the rest are easy blocks. This convention is followed throughout the chapter. The key



(a) Workload schedule on CPU and GPU (b) # Hard blocks v.s. latency

Figure 4.2: (a) Workload scheduling between CPU and GPU and (b) relationship between number of hard blocks and execution latency.

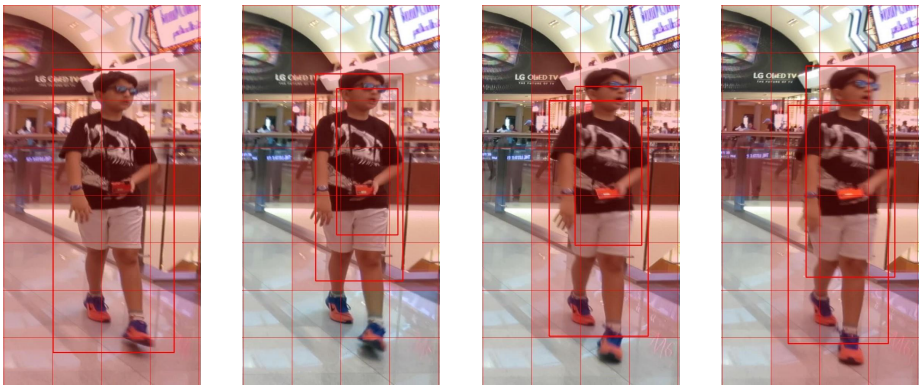


Figure 4.3: Example of block artifacts.

frame is fully processed and all block features are cached. For the following frames, hard blocks are processed to generate new block features, which are then combined with the cached easy block features to produce the bboxes. Clearly, even though the differences between these frames are tiny, the feature merging still introduces inconsistency at the boundaries between hard and easy blocks, resulting in inaccurate and noisy bboxes.

4.3 BlockHybrid Design

In this section, we present the design of BlockHybrid, including the system overview, block decision-making with the policy network, post-processing to remove redundant bboxes, and block-wise fine-tuning to mitigate block artifacts.

4.3.1 Overview

BlockHybrid is an edge video analytics framework designed for object detection tasks. It employs hybrid block-wise conditional execution to reduce computation and communication overhead. The core of BlockHybrid is its policy network, which outputs suitable actions for each block. Unlike supervised learning, where labels for hard and easy blocks are fixed and cannot adapt to varying trade-offs, a reinforcement learning approach can balance performance and efficiency based on the specific requirements of the task using judiciously designed reward functions.

BlockHybrid comprises a camera and an edge server. The camera performs three tasks: 1) capturing video frames and communicating with the server, 2) running the tracker, and 3) executing the policy network. The edge server hosts a detector to process frames or blocks from the camera. We adopt the CUDA operators from [19] to perform block-wise detection tasks efficiently. As explained in Section 2.2, a standard detector can process frames in blocks and generate object bboxes, facilitated by these operators.

Figure 4.4 depicts the workflow of BlockHybrid. The first frame of every T frames is marked as a key frame and the rest are marked as regular frames ①. BlockHybrid operates in two distinct phases: full execution (F) and partial execution (P), applied to key frames and regular frames, respectively. The workload scheduling between the

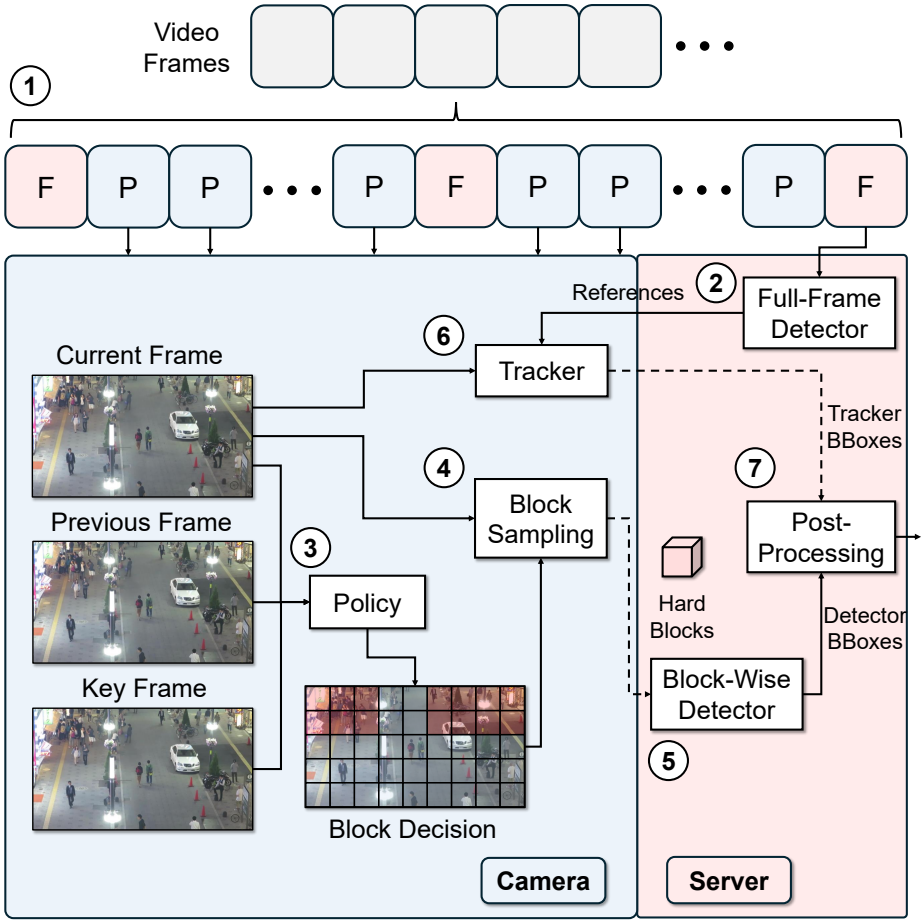


Figure 4.4: System overview of BlockHybrid.

camera and server in both phases is illustrated in Figure 4.5. In the full execution phase, a key frame is sent to the edge server, where a detector will process the full frame, generate references and cache the block features ②, as illustrated in Figure 4.6. The references are then sent back to the camera. In the partial execution phase, the policy network ③ takes the current frame, previous frame and key frame as inputs, and determines hard and easy blocks. Based on the decision, block sampling ④ extracts hard blocks from the current frame. The hard blocks are then transmitted to the server and processed by the detector ⑤. Only the hard blocks are executed, while the

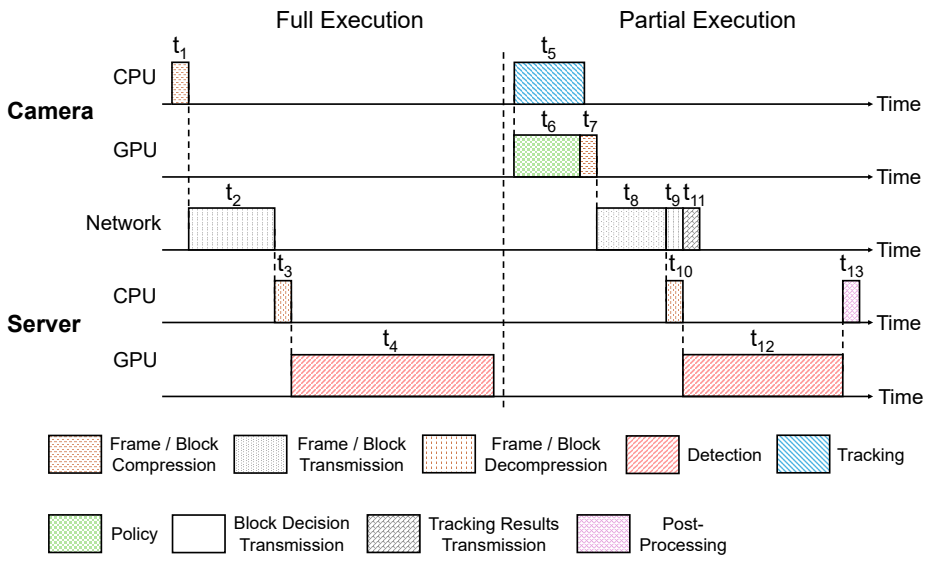


Figure 4.5: Workload scheduling between camera and server. Data migration time (e.g., from CPU to GPU) is omitted considering its negligible overhead.

features of other blocks (i.e., easy blocks) are retrieved from the local cache, which is updated every key frame. The features of both hard and easy blocks are merged into complete maps—a processing step called *block merge*, which is performed at each convolutional layer in the detector. Due to this mechanism, the hard blocks, being freshly executed, produce accurate bboxes, while those generated from easy blocks might be outdated as they rely on cached features from key frame executions. During the process of ③, ④ and ⑤, the tracker ⑥ tracks all references within key frames to generate tracking results. The location and size of each object are tracked, and if they change significantly (e.g., exceeding a defined threshold), the tracking result is deemed unreliable, and a Kalman filter is used to make estimations instead. As shown in Figure 4.5, the detector and the tracker run in parallel. The execution of the tracker will not incur additional overhead as long as $t_5 \leq t_6 + t_7 + t_8 + t_{10} + t_{12}$. Lastly, the detection and tracking bboxes go through the post-processing stage ⑦,

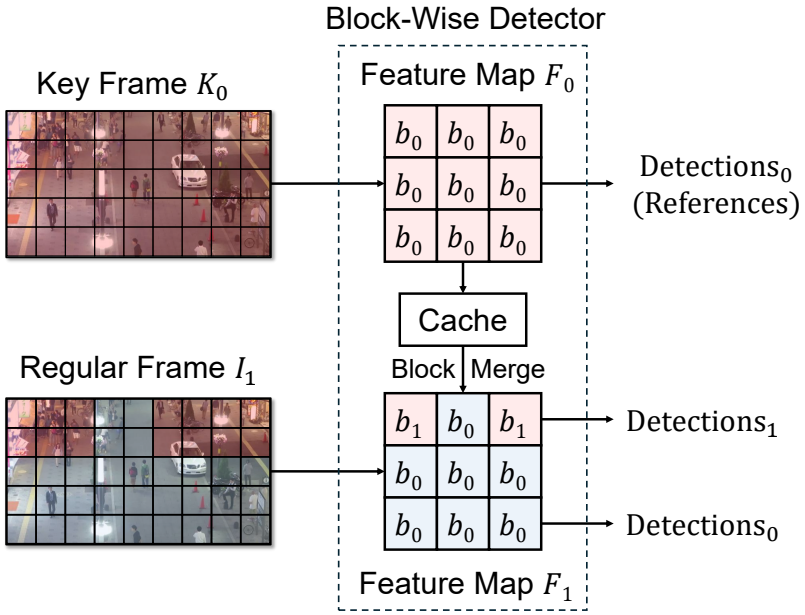


Figure 4.6: Brief process of block-wise detection.

where redundant bboxes are eliminated. Note that the full-frame detector and the block-wise detector are essentially the same detector. The detection mode can be switched based on the input type. We separate them here for better clarity.

4.3.2 Policy Network

The policy network f_{pn} with parameters θ, is a trainable convolutional neural network that can make binary *block decisions* (i.e., whether is a block is hard or easy) for the current frame at timestamp t, based on the state:

$$\mathcal{S}_t = \left\{\mathcal{I}_t, \mathcal{I}_{t-1}, \mathcal{K}^T\right\}, \tag{4.3.1}$$

where $\mathcal{I}_t$, $\mathcal{I}_{t-1}$ and $\mathcal{K}^T$ are the current frame, previous frame, and key frame (updated every T frames), respectively. The network outputs a probabilities mask:

$$\mathcal{P}_t = f_{pn}\left(S_t; \theta\right), \tag{4.3.2}$$

containing probabilities p_b for each block b:

$$\mathcal{P}_t = [p_1, \ldots, p_b, \ldots, p_B] \in [0,1]^B, \tag{4.3.3}$$

where B is the total number of blocks. The soft probabilities are sampled based on Bernoulli distribution to binary actions:

$$\mathcal{A}_t = [a_1, \ldots, a_b, \ldots, a_B] \in \{0,1\}^B, \tag{4.3.4}$$

where $a_b \sim \text{Bernoulli}(p_b)$. When $a_b = 1$, block b is processed by the block-wise detector; otherwise, it is handled by the light tracker.

Training: The policy network is trained offline with ground truth bboxes. The policy π predicts actions $\mathcal{A}$ (omitting t for simplicity) with the goal of maximizing the reward per frame. The objective can be represented as:

$$\max \mathcal{J}\left(\theta\right) = \max \mathbb{E}_{\mathcal{A}\sim\pi_\theta}\left[\mathcal{R}\left(\mathcal{A}\right)\right], \tag{4.3.5}$$

where the total reward $\mathcal{R}(\mathcal{A})$ is defined as the sum of all the block rewards in the frame:

$$\mathcal{R}\left(\mathcal{A}\right) = \sum_{b=1}^{B}\left[\mathcal{R}\left(a_b\right)\right]. \tag{4.3.6}$$

Therefore, Eq. (4.3.5) can be written as:

$$\max \mathcal{J}(\theta) = \max \sum_{b=1}^{B} \left(\mathbb{E}_{a_b \sim \pi_{b,\theta}} \left[\mathcal{R}_b(a_b) \right] \right). \tag{4.3.7}$$

The policy network's parameters θ can be updated using gradient ascent with learning rate α:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \left[\mathcal{J}(\theta) \right]. \tag{4.3.8}$$

Based on [19], maximizing the objective function in Eq. (4.3.7) is equivalent to minimizing the following loss function:

$$\mathcal{L} = -\sum_{b=1}^{B} \left(\mathcal{R}_b(a_b) \log \pi_{b,\theta}(a_b \mid \mathcal{S}_t) \right). \tag{4.3.9}$$

Reward: The purpose of the policy network is to identify the hard blocks for detector processing. To prevent the policy from converging to a suboptimal state where it always chooses to process all blocks as hard blocks, the reward takes both accuracy and computation cost into account:

$$\mathcal{R}_b(a_b) = \mathcal{R}_{TE}(a_b) + \gamma \mathcal{R}_{cost}(a_b), \tag{4.3.10}$$

where $\mathcal{R}_{TE}(a_b)$ is the accuracy reward based on task error, $\mathcal{R}_{cost}(a_b)$ is the reward for computation cost, and γ is a hyperparameter for balancing both rewards.

Task error: To determine the *hardness* of a block, we define *task error* (TE) as a measure of the error resulting from processing that block either using the detector or tracker, compared to the ground truth. Simply put, task error reflects how much a predicted bbox deviates from the ground truth, which represents the actual position

of an object. Algorithm 1 is used to compute the task error. It requires the pipeline execution results $\mathcal{B}$ (including bboxes from both the detector and the tracker), the ground truth bboxes $\mathcal{G}$, and the detection results $\mathcal{D}$, which are derived from executing full frames with the detector. The task error is first initialized as a zero-filled matrix with the same size as the input frame. Next, undetectable ground truth bboxes are removed. These bboxes cannot be detected by the detector and, consequently, by BlockHybrid. Therefore, they are excluded from the calculation of TE. The rationale is that if BlockHybrid cannot predict these bboxes, processing the corresponding blocks is unnecessary. Then, for each bbox in $\mathcal{B}$, we use greedy matching to find the best matched bbox in the updated ground truth $\mathcal{G}'$, and measure the task error based on IoU. Large overlap indicates low task error, and potentially low block hardness. Finally, for unmatched ground truth bboxes, i.e., false negatives, we set the task error to the maximum value (i.e., 1.0). The returned matrix TE is pixel-wise; we convert it to block-wise representation using max-pooling:

$$TE_b = \max TE_p \quad \forall p \in b. \tag{4.3.11}$$

The block-wise accuracy reward $\mathcal{R}_{TE}(a_b)$ is finally calculated as:

$$\mathcal{R}_{TE}(a_b) = \begin{cases} TE_b & \text{if } a_b = 1, \\ -TE_b & \text{if } a_b = 0. \end{cases} \tag{4.3.12}$$

Hard blocks, where $a_b = 1$, have a positive reward for positive task error. In contrast, easy blocks (i.e., $a_b = 0$) receive a negative reward, and high task error increases the likelihood of being decided as hard blocks.

Computation cost: As mentioned in Section 4.3.1, the detector and tracker are executed in parallel. Since the detector is more computationally expensive, the overall processing time is dominated by the detector. Therefore, the computation cost of a frame is measured by the number of detector-processed blocks (i.e., $a_b = 1$):

$$\mathcal{C} = \frac{\sum_{i=1}^{B} a_i}{B} \in [0, 1]. \tag{4.3.13}$$

Then, we define the cost reward as:

$$\mathcal{R}_{cost}(a_b) = \begin{cases} \tau - \mathcal{C} & \text{if } a_b = 1, \\ -(\tau - \mathcal{C}) & \text{if } a_b = 0, \end{cases} \tag{4.3.14}$$

where τ is an execution target, defining the desired average cost. The reward minimizes the difference between $\mathcal{C}$ and the desired percentage τ. If the cost $\mathcal{C}$ is below the target τ, the policy receives a positive reward, encouraging it to process more hard blocks. Otherwise, it is penalized with a negative reward, promoting the processing of more easy blocks. Since the policy network is trained offline with ground truth, we follow the approach introduced in Section 4.2.1 to estimate the number of hard blocks in each training sample beforehand. Finally, the target τ is calculated as the average number of hard blocks among T frames:

$$\tau = \frac{\sum_{i=1}^{T} H_i}{T} \in [0, 1], \tag{4.3.15}$$

where H is the estimated number of hard blocks.

Algorithm 1 Task Error for Object Detection

Require: execution results $\mathcal{B}$, ground truth $\mathcal{G}$, detection results $\mathcal{D}$, IoU threshold δ
// Initialize task error as a zero matrix of size $H \times W$
$TE \leftarrow 0^{H \times W}$
// remove undetectable ground truth bboxes
$\mathcal{G}' \leftarrow \varnothing$
for $gt \in \mathcal{G}$ **do**
  $IoU_{best} \leftarrow 0$
  for $det \in \mathcal{D}$ **do**
    if IoU $(gt, det) > IoU_{best}$ **then**
      $IoU_{best} \leftarrow$ IoU (gt, det)
    end if
  end for
  if $IoU_{best} \geq \delta$ **then**
    $\mathcal{G}' \leftarrow \mathcal{G}' \cup gt$
  end if
end for
// measure pixel-wise task error
for $bbox \in \mathcal{B}$ **do**
  $IoU_{best} \leftarrow 0$
  $gt'_{best} \leftarrow NULL$
  for $gt' \in \mathcal{G}'$ **do**
    if IoU $(gt', bbox) > IoU_{best}$ **then**
      $IoU_{best} \leftarrow$ IoU $(gt', bbox)$
      $gt'_{best} \leftarrow gt'$
    end if
  end for
  if $gt'_{best} \neq NULL$ **then**
    $\mathcal{G}' \leftarrow \mathcal{G}' \setminus \{gt'_{best}\}$
    for all pixels $p \in gt'_{best}$ **do**
      $TE_p \leftarrow \max(TE_p, 1 - IoU_{best})$
    end for
  end if
end for
// deal with unmatched ground truth bboxes
for $gt' \in \mathcal{G}'$ **do**
  for all pixels $p \in gt'$ **do**
    $TE_p \leftarrow 1.0$
  end for
end for
return TE

4.3.3 Post-Processing

As introduced in Section 4.3.1, detection bboxes are generated from both hard and easy blocks. Meanwhile, the tracker tracks all references from key frames, producing tracking bboxes that also span hard and easy blocks. The purpose of post-processing is to refine these results by removing detection bboxes in easy blocks (i.e., stale detections) and tracking bboxes in hard blocks (i.e., less accurate ones). However, this refinement process is challenging since some bboxes may span across block boundaries, making it difficult to determine their associated blocks. To address this issue, we propose a two-stage filtering algorithm. In the first stage, the algorithm evaluates each bbox's area within different block types. Detection bboxes are retained if more than 50% of their area overlaps with hard blocks, while tracking bboxes are kept if more than half of their area falls within easy blocks. The rest are simply discarded. This stage ensures that bboxes are appropriately assigned to their respective block types. In the second stage, a non-maximum suppression (NMS) algorithm is applied to further eliminate overlapping bboxes.

4.3.4 Block-wise Fine-tuning

The combination of new and old block features during block merge potentially leads to feature inconsistencies at block boundaries. This issue, known as block artifacts, can result in inaccurate or redundant detections. To mitigate this problem, we propose block-wise fine-tuning, which is an additional training stage applied to pre-trained weights of the detector. During fine-tuning, for each training iteration, the model is given a pair of adjacent frames $\{\mathcal{I}_t, \mathcal{I}_{t+1}\}$ to simulate our key frame and regular frame inference procedure. For $\mathcal{I}_t$, the detector processes the entire frame and caches all

block features. For $\mathcal{I}_{t+1}$, ground truth bboxes are used to identify activated blocks (i.e., blocks to be executed), among which $\eta = 30\%$ of the blocks are turned into inactive blocks, to simulate scenarios where objects are located at block boundaries. The execution results are compared with the ground truth to calculate the loss, which is then used to update the model weights, improving its ability to handle block artifacts.

4.4 Evaluation

In this section, we conduct experiments to evaluate the performance of BlockHybrid. We start with the experimental setup and then compare BlockHybrid to baselines on two benchmark datasets and a real-world testbed.

4.4.1 Experimental Setup

The hardware platform consists of the NVIDIA Jetson TX2 and the Dell desktop mentioned in Section 2.5. In benchmark evaluation, all workloads are executed locally on the desktop. In testbed evaluation, the Jetson TX2 serves as a smart camera, while the desktop acts as an edge server, with workloads distributed between them, as shown in Figure 4.5. The edge server and camera are connected with the D-Link AX4800 router through a 2.4GHz WiFi network. The bandwidth is 40.5 Mbps, measured by `iperf`. TCP is used to enable the communication between the camera and the server. Frames and blocks are compressed in JPEG format using the OpenCV library [40, 95] before transmission, and decompressed by the server upon reception for further processing.

4.4.2 Datasets and Metrics

BlockHybrid is evaluated using the MOT17 and WildTrack datasets described in Section 2.3. Each video sequence is divided into 50% for detector training, 25% for policy network training, and 25% for testing. All frames are resized to 1024×2048. We adopt mAP@0.5 to evaluate accuracy, and use end-to-end latency and network traffic to evaluate efficiency, as introduced in Section 2.4.

4.4.3 BlockHybrid Configuration

To demonstrate the versatility of BlockHybrid, we apply it to two types of block-wise object detectors: 1) CNN-based and 2) transformer-based. For simplicity, we refer to them as BlockHybrid (CNN) and BlockHybrid (Transformer). For BlockHybrid (CNN), we adopt CSP [154] as the detector framework, with ResNet-50 [43] or MobileNet [155] as backbone (denoted as CSP + ResNet-50/MobileNet). For BlockHybrid (Transformer), we use Faster-RCNN + ViT-small. The network configuration follows that of [96]. Specifically, encoder depth $L = 12$, embedding dimension $D = 384$, patch size $= 16 \times 16$. The pre-trained weights of ResNet-50 and ViT-Small are from [156] and [157], respectively, while those of MobileNet are from official Pytorch repository. For detector training, we follow [156] for CSP and [158] for Faster-RCNN, using the same training hyperparameters as in their respective frameworks. The hyperparameters for block-wise fine-tuning are consistent with those used for detector training. The policy network employs ResNet-8 as its backbone and three 64-channel convolutional layers as its head. The block size is set to 128×128. To ensure seamless integration with transformer-based detectors, where the patch size is 16×16, we define one block as equivalent to 8×8 patches. The policy network is

trained offline using an RMS optimizer with learning rate = $1e^{-4}$ and weight decay = $1e^{-3}$. The key frame interval $T = 10$, and the balancing hyperparameter $\gamma = 5$.

4.4.4 Baselines

BlockHybrid is compared with the following baselines. As discussed in Section 2.2, EHCI [95] and BlockCopy [19] represent the SOTA methods in two different categories of block-wise conditional execution, both employing CNN-based detectors. Arena [96] serves as the SOTA method for transformer-based block-wise conditional execution. For fairness, we compare BlockHybrid (CNN) with BlockCopy and EHCI, and BlockHybrid (Transformer) with Arena. In each comparison, all baselines use the same detector and backbone, while block selection follows their respective original designs.

- **Full-frame detector (FD):** The camera always transmits full frames to the edge server, where a base detector performs inference without employing any additional techniques. FD uses CSP when compared with BlockHybrid (CNN), and Faster-RCNN when compared with BlockHybrid (Transformer).

- **BlockCopy [19]:** A policy network is used to determine informative blocks on the camera, which are then sent to the server and batch-processed by the detector. The features of non-informative blocks are cached and updated in the server. We set the execution target τ to 30%.

- **EHCI [95]:** Informative blocks are identified on the camera based on detection results from the previous frame. These non-uniform blocks are sent to the server, where they are arranged into a compact frame using a rectangle packing

algorithm (e.g., Next-Fit [95]) before performing frame-wise object detection. The detection results are subsequently mapped back to their original locations in the full frame.

- **Arena [96]:** Every T frames, a full frame is sent to the server for processing and the resulting tokens are cached. For the remaining $T-1$ frames, detection results from the previous frame are used to decide informative patches, which are then transmitted to the server for processing. The encoder processes only these informative patches, while the decoder reconstructs the complete token sequence by reusing cached tokens from prior executions.

4.4.5 Benchmark Evaluation

In benchmark evaluation, all methods are executed locally on the edge server. Tables 4.2 and 4.3 present results using CSP + ResNet-50 and MobileNet. As shown in Table 4.2, BlockHybrid (CNN) achieves the highest FPS among all the baselines, with the least number of blocks being executed by the detector. Meanwhile, its accuracy remains comparable to the full-frame detector, with only a small drop of $\sim$1%. In contrast, BlockHybrid*, without block-wise fine-tuning, struggles to predict accurate bboxes across block boundaries, leading to a significant accuracy drop of $\sim$5%. Compared to BlockHybrid, BlockCopy shows a lower FPS since it tends to process the number of blocks defined by the execution target $\tau = 30\%$. However, processing only $\sim$30% of the blocks sometimes misses some informative regions, resulting in a noticeable accuracy decrease of $\sim$2%. EHCI, on the other hand, attempts to process all informative blocks, which significantly reduces its FPS. As its block size is non-uniform, the number of blocks is calculated by dividing the area of the merged smaller

frame by 128×128, the block size used in BlockHybrid and BlockCopy. Table 4.3 shows similar trends to Table 4.2. Notably, BlockHybrid with MobileNet achieves near real-time processing (i.e., ∼30 FPS), representing a significant improvement over FD, which achieves only ∼16 FPS.

Table 4.4 presents results using Faster-RCNN + ViT-Small. The overall trends remain the same: BlockHybrid (Transformer) significantly reduces the number of processed blocks, improving FPS while maintaining accuracy close to FD. However, compared to CNN-based methods, all ViT-based methods run at a lower FPS due to the increased computation overhead of self-attention operations. Arena, despite being optimized for transformer-based block-wise execution, still suffers from reduced efficiency since it processes a relatively large portion of the frame. In contrast, BlockHybrid effectively balances computation efficiency and detection performance, achieving up to ∼4.7 FPS with a small accuracy drop (<2%) compared to FD.

We also study the impact of different key frame intervals T on the trade-off between accuracy and the number of executed hard blocks, as shown in Figure 4.7. In this experiment, BlockHybrid (CNN) adopts CSP + ResNet-50. When $T = 5$, BlockHybrid achieves the highest accuracy, with around 30% of the blocks being executed as hard blocks for MOT17 and WildTrack. When $T = 10$, BlockHybrid reduces the number of hard blocks by 19.5% with only a 0.4% accuracy loss for MOT17, and by 14.0% at the expense of a 0.5% accuracy drop for WildTrack. As T increases further from 15 to 30, the number of hard blocks stabilizes or slightly increases. This is because the references gradually become outdated, leading to more inaccurate tracking results, which in turn activates more hard blocks by the policy network. Meanwhile,

Table 4.2: Benchmark evaluation on two datasets using CSP + ResNet-50.

Dataset	Method	Backbone	Accuracy	Blocks	FPS
MOT17	BlockHybrid	ResNet-50	78.0%	**23.1%**	**9.5**
	BlockHybrid*	ResNet-50	73.1%	25.6%	9.1
	FD	ResNet-50	**79.0%**	100.0%	3.4
	BlockCopy	ResNet-50	77.2%	39.2%	8.0
	EHCI	ResNet-50	77.8%	45.6%	6.8
WildTrack	BlockHybrid	ResNet-50	72.5%	**25.1%**	**9.2**
	BlockHybrid*	ResNet-50	67.2%	28.1%	8.8
	FD	ResNet-50	**73.4%**	100.0%	3.4
	BlockCopy	ResNet-50	71.6%	39.2%	8.0
	EHCI	ResNet-50	72.1%	49.8%	6.3

[1] BlockHybrid* denotes without block-wise fine-tuning.

Table 4.3: Benchmark evaluation on two datasets using CSP + MobileNet.

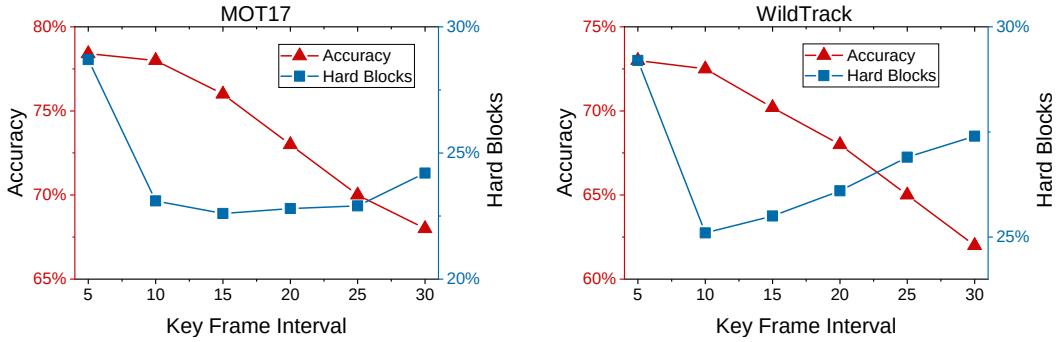
Dataset	Method	Backbone	Accuracy	Blocks	FPS
MOT17	BlockHybrid	MobileNet	72.2%	**23.8%**	**27.2**
	BlockHybrid*	MobileNet	67.0%	26.9%	26.0
	FD	MobileNet	**72.7%**	100.0%	15.9
	BlockCopy	MobileNet	71.0%	39.7%	24.4
	EHCI	MobileNet	71.9%	44.7%	21.3
WildTrack	BlockHybrid	MobileNet	65.8%	**25.9%**	**26.2**
	BlockHybrid*	MobileNet	60.1%	29.5%	25.1
	FD	MobileNet	**66.5%**	100.0%	15.8
	BlockCopy	MobileNet	64.6%	40.2%	23.9
	EHCI	MobileNet	65.4%	48.5%	20.6

[1] BlockHybrid* denotes without block-wise fine-tuning.

Table 4.4: Benchmark evaluation on two datasets using Faster-RCNN + ViT-Small.

Dataset	Method	Backbone	Accuracy	Blocks	FPS
MOT17	BlockHybrid	ViT-Small	80.1%	**23.4%**	**4.7**
	BlockHybrid*	ViT-Small	78.3%	25.1%	4.5
	FD	ViT-Small	**81.5%**	100.0%	1.5
	Arena	ViT-Small	80.6%	43.9%	3.5
WildTrack	BlockHybrid	ViT-Small	74.5%	**24.5%**	**4.6**
	BlockHybrid*	ViT-Small	72.1%	26.6%	4.4
	FD	ViT-Small	**76.3%**	100.0%	1.5
	Arena	ViT-Small	75.0%	47.3%	3.3

[1] BlockHybrid* denotes without block-wise fine-tuning.



(a) MOT17 (b) WildTrack

Figure 4.7: The influence of different key frame intervals on the trade-off between accuracy and the number of hard blocks.

accuracy shows a noticeable decline as the cached block features become stale, exacerbating block artifacts and degrading performance. To summarize, $T = 10$ achieves the best balance between accuracy and the number of hard blocks across both datasets. In practice, it is non-trivial to identify the optimal T, since the ground truth is unavailable during the online stage. However, this problem can be solved by offline or online profiling [10], which we leave for future endeavors.

4.4.6 Testbed Evaluation

In testbed evaluation, we use CSP + MobileNet for all methods. Figure 4.8 shows the normalized network traffic and accuracy of different methods across two datasets. Figure 4.9 provides the end-to-end latency for each method, broken down into camera time, transmission time, and server time. Due to minor information loss during frame compression, the accuracy of all approaches experiences a slight drop of $< 2\%$, while their relationship remains consistent with the results in Table 4.3. Since BlockHybrid utilizes parallel computation, as shown in Figure 4.5, the camera processing time overlaps with the communication time, and the latter partially overlaps with the

server time. The camera time dominates the end-to-end latency of BlockHybrid, since it includes the overhead of running the policy network and the tracker on a resource-poor device. However, the reduction in the number of processed blocks leads to significantly lower communication and server computation costs, enabling BlockHybrid to achieve the lowest end-to-end latency among all the approaches. FD, which always transmits the full frame to the server, generates the most network traffic and thus incurs the longest end-to-end latency, with the transmission time being the dominating factor. As shown in Table 4.3, BlockCopy and EHCI transmit and process more blocks than BlockHybrid, leading to higher transmission time and server time. Overall, BlockHybrid achieves the best accuracy-latency trade-off, accelerating the end-to-end processing by 31.5%–39.1%, with only a minimal accuracy drop of 0.7%–1.3%, compared to FD.

To further enhance system efficiency, we apply inter-frame pipelining, which allows different processing stages: camera processing (L_C), transmission (L_T), and server-side processing (L_S) of different frames, to overlap in time. The average per-frame latency can be approximated as $L_{\text{avg}} \approx \max(L_C, L_T, L_S)$, meaning that the overall pipeline is bottlenecked by the slowest stage [159]. As shown in Figure 4.9 and Figure 4.10, the bottleneck varies depending on the method. For BlockHybrid and BlockCopy, the policy network runs on a resource-constrained embedded device, making camera-side computation the bottleneck. In contrast, FD suffers from transmission bottlenecks, as it sends full frames to the server, leading to significantly higher communication overhead. Meanwhile, EHCI processes more blocks than BlockCopy, shifting the bottleneck to server-side computation. Overall, with pipelining, BlockHybrid still achieves the lowest per-frame latency among all methods, demonstrating its

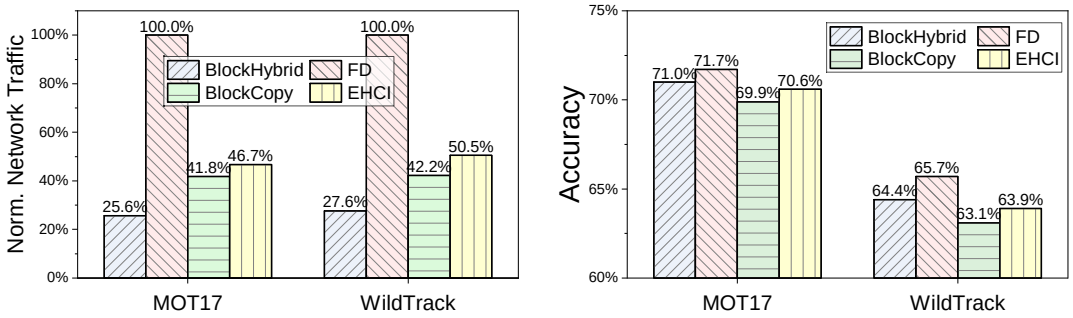
effectiveness in latency-sensitive edge video analytics applications. This experiment also highlights the importance of carefully balancing the computation and communication costs across different pipeline stages to maximize system throughput.

4.4.7 Visualization

To further demonstrate the performance of BlockHybrid, we provide visualizations of three scenes in Figure 4.11, with the number of hard blocks indicated below each image. It is evident that the policy network effectively differentiates between hard and easy blocks. The easy blocks are mainly found in sparse regions with few objects, whereas the hard blocks concentrate in crowded regions, regions with occluded objects, and regions with incoming objects, aligning well with expectations.

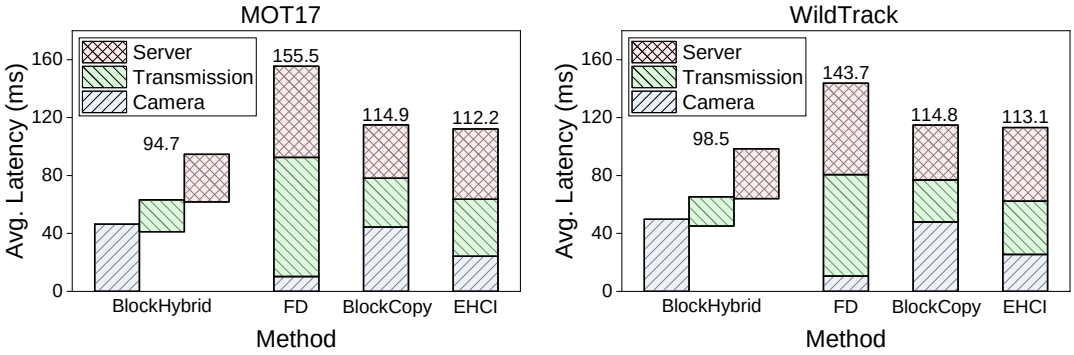
4.5 Conclusion

In this chapter, we introduced BlockHybrid, a novel framework for efficient object detection that leverages fine-grained block-wise conditional execution to mitigate the inefficiencies of traditional pipelines. By classifying blocks as hard or easy using a policy network and applying different processing strategies accordingly, BlockHybrid significantly reduces redundant computation and communication. Extensive evaluations on public benchmarks and a real-world testbed demonstrate that BlockHybrid achieves a superior trade-off between accuracy and efficiency, outperforming SOTA methods.



(a) Normalized network traffic (b) Accuracy

Figure 4.8: (a) Normalized network traffic and (b) accuracy of different methods on two datasets.



(a) MOT17 (b) WildTrack

Figure 4.9: Average end-to-end latency of different methods on two datasets. End-to-end latency includes camera time, transmission time and server time.

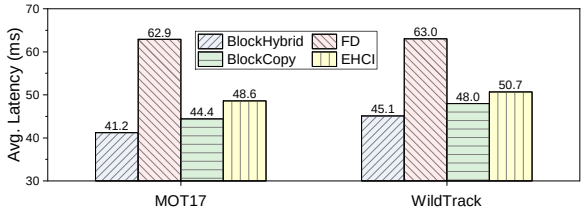


Figure 4.10: Average end-to-end latency of different methods with pipelining on two datasets.

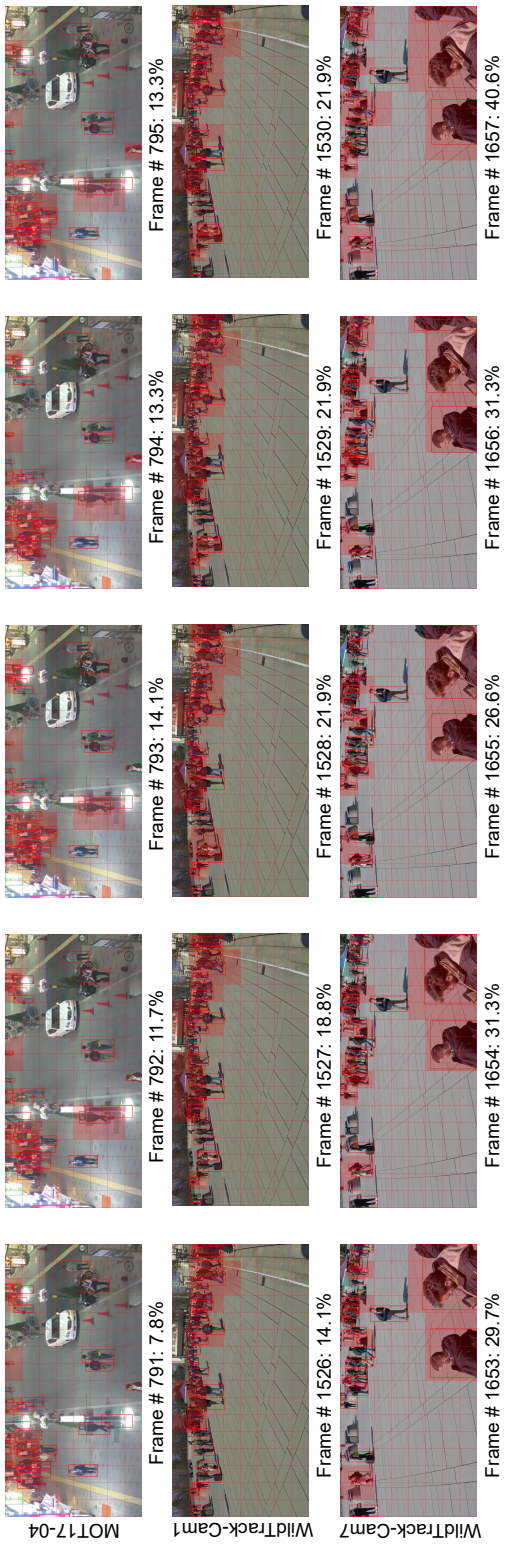


Figure 4.11: Visualization of BlockHybrid across three scenes.

Chapter 5

SEED: An End-to-End Selective Execution Framework for Transformer-Based Object Detection in Edge Video Analytics

This chapter is based on the manuscript: Renjie Xu and Rong Zheng, "SEED: An End-to-End Selective Execution Framework for Transformer-Based Object Detection in Edge Video Analytics", which is currently under submission. The work has not been published, and copyright remains with the author at the time of thesis submission.

5.1 Introduction

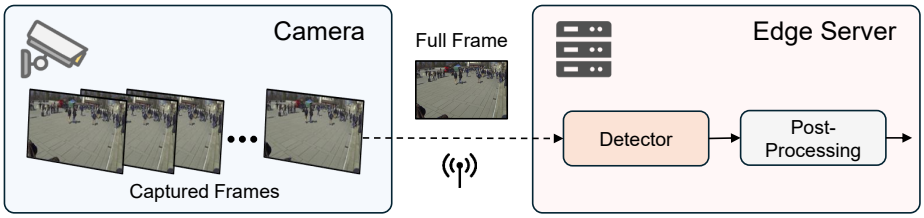
Cameras have become deeply embedded in the urban environments, driven by the rapid development of IoT and visual sensing technologies. They are now ubiquitous in streets, buildings, factories, and homes, supporting a broad range of smart applications. As a result, video analytics has become a central focus in both research and industry, enabling capabilities such as traffic monitoring, anomaly detection, and behavior analysis [2]. Many of these applications demand low-latency or even real-time processing. For example, in ITS [33], promptly detecting abnormal driving or pedestrian behavior is critical for issuing early warnings to drivers via vehicular communication networks, helping to prevent potential traffic accidents. In such latency-sensitive scenarios, it is crucial to design an efficient execution framework that enables rapid and accurate video analytics.

Recently, ViTs [160] have gained popularity for their strong representation capability, outperforming CNNs in many video analytics tasks [21, 161]. However, their self-attention mechanism incurs high computational cost due to its quadratic complexity with respect to token count [160]. For instance, a 1024×2048 frame with 16×16 patch size produces over 8000 tokens, posing significant challenges for deployment in edge video analytics systems [96]. To mitigate this, prior work explores network pruning [22], parameter quantization [23], layer splitting [162], and efficient attention [24, 25], aiming to reduce overhead while preserving accuracy.

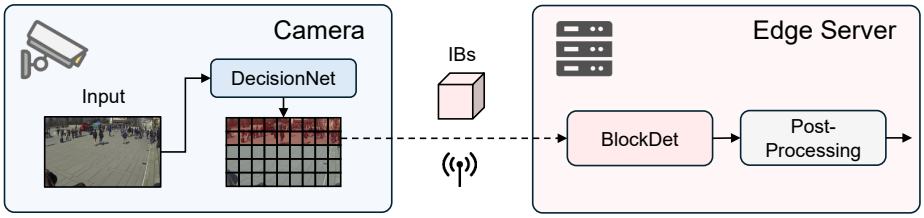
However, most existing methods are content-agnostic and treat all regions in video frames equally. In practice, it is often unnecessary to process entire frames uniformly. For example, traffic monitoring primarily concerns dynamic entities like vehicles and pedestrians, instead of background regions [33]. This motivates *selective execution*,

which allocates computation to semantically important regions while reducing or skipping others. Final predictions are generated by merging features from heterogeneous regions. Selective execution is widely adopted in edge video analytics systems [2], where a smart camera selects informative regions at runtime and transmits them to an edge server for further processing. Prior works [95, 94, 96] estimate informative regions by expanding past detections using motion heuristics, often leading to over-selection beyond necessary scope. Others [19, 152] use policy networks that are not jointly optimized with the detector. As a result, the detector is not adapted to the selected blocks, which may lead to feature inconsistencies at region boundaries and compromise accuracy. Token pruning methods [101, 100] adopts gating modules embedded within the detector to select informative tokens. This coupling limits their use in edge deployments where block selection must be performed upfront to filter out non-IBs before transmission.

To mitigate these weaknesses, we propose SEED, an end-to-end selective execution framework for accelerating ViT-based object detection pipelines. SEED consists of a lightweight decision network (DecisionNet) and a block-wise ViT detector (BlockDet). Each frame is divided into uniform-sized blocks; DecisionNet identifies informative blocks (IBs) using semantic cues. IBs are fully processed by BlockDet, while non-IBs are lightly processed. To demonstrate the flexibility of SEED, we design two variants with distinct selective execution strategies: SEED-TR (token reuse), which reuses historical features for non-IBs, and SEED-EE (early exit), which terminates inference early for non-IBs. The DecisionNet and BlockDet are jointly trained in an end-to-end manner, optimizing selection strategies directly for detection performance. However, this is non-trivial due to unstable block decisions and poor detection quality



(a) Conventional pipeline



(b) Proposed pipeline

Figure 5.1: Comparison between conventional pipeline and the proposed pipeline

.

at early stages. To address this, we propose a three-stage training strategy involving BlockDet pre-training, DecisionNet warm-up, and joint optimization. In the last two stages, pseudo-labels derived from ground-truth annotations are used to supervise DecisionNet, making the approach detector-agnostic. Moreover, the DecisionNet is designed to be lightweight and can run efficiently on resource-constrained IoT devices (e.g., smart cameras). This enables a distributed deployment, where the DecisionNet runs on the camera side to select IBs that are then sent to the BlockDet deployed on an edge server for processing. As illustrated in Figure 5.1b, only the selected IBs are transmitted to the server, significantly reducing communication and computational overhead compared to naively transmitting full frames, as shown in Figure 5.1a.

To evaluate the performance of SEED, we compare it against state-of-the-art approaches on two public datasets and a real-world edge video analytics testbed, using an NVIDIA Jetson TX2 as the camera and an Ubuntu desktop as the edge

server. Experimental results demonstrate that SEED significantly accelerates end-to-end processing while maintaining competitive accuracy.

In summary, our key contributions are as follows:

- We propose SEED, an end-to-end selective execution framework that prioritizes computation on IBs to accelerate ViT-based detection.
- We design a lightweight and decoupled DecisionNet for real-time block selection.
- We implement two SEED variants: SEED-TR (token reuse) and SEED-EE (early exit) to demonstrate flexible execution strategies.
- We present a three-stage training strategy with pseudo-label supervision, enabling stable and joint optimization of DecisionNet and BlockDet.
- We deploy a prototype of SEED on a real-world testbed, validating its performance in edge video analytics.

5.2 Motivation

In this section, we demonstrate the spatial redundancy in video frames and the potential efficiency gains enabled by selective execution.

5.2.1 Redundancy of Videos

Videos often contain significant spatial redundancy, leading to additional overhead when all regions are processed uniformly. To quantify this redundancy, we analyze two representative datasets: MOT17 [104] and WildTrack [105], each comprising multiple video sequences. Each frame is resized to 1024×2048 and divided into 16×32

Table 5.1: Redundancy of MOT17 and WildTrack datasets.

Sequence	**# Frame**	**Density**[1]	**IBs**[2]
MOT17-02	600	31.0	22.5%
MOT17-04	1050	45.3	35.6%
MOT17-05	837	8.3	72.2%
MOT17-09	525	10.1	29.8%
MOT17-10	654	19.6	19.6%
MOT17-11	900	10.5	42.7%
MOT17-13	750	15.5	8.0%
WildTrack-Cam1	2000	35.8	27.2%
WildTrack-Cam2	2000	24.9	42.9%
WildTrack-Cam3	2000	38.3	47.6%
WildTrack-Cam4	2000	29.3	22.9%
WildTrack-Cam5	2000	27.0	31.0%
WildTrack-Cam6	2000	34.2	27.8%
WildTrack-Cam7	2000	30.9	30.5%

[1] Density denotes the average number of objects per frame.
[2] IB refers to informative block.

blocks of size 64×64. These sequences span a wide range of crowd densities, with object counts per frame ranging from 8.3 to 45.3 in MOT17 and from 24.9 to 38.3 in WildTrack. Despite such variations in scene complexity, the proportion of IBs, defined as the blocks containing target objects, remains relatively low across most sequences.

As shown in Table 5.1, several sequences in MOT17, such as MOT17-02 and MOT17-13, exhibit extremely sparse distributions of IBs, covering only 22.5% and 8.0% of the spatial grid, respectively. Even in denser scenarios like MOT17-04, IBs still only account for 35.6%. Similar trends are observed in WildTrack, where the IB ratio stays below 50% in most camera views, reaching as low as 27.2% in Cam1. These statistics highlight the substantial irrelevant or redundant spatial content present in

real-world videos.

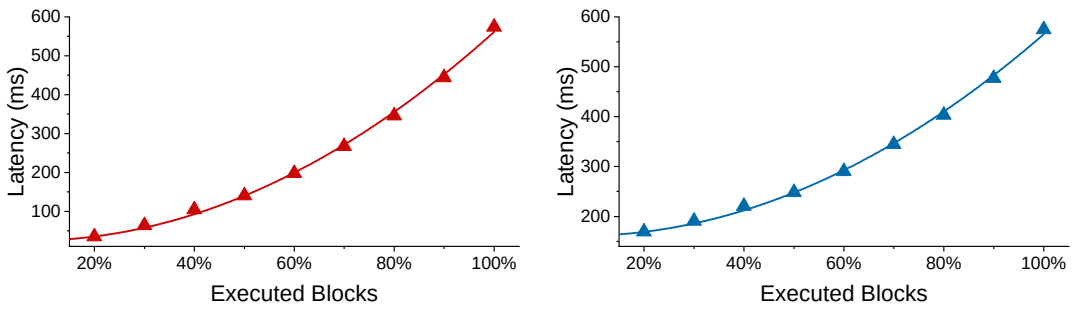
5.2.2 Benefits of Selective Execution in ViTs

For a single ViT encoder with N input tokens and token dimension D, the per-layer complexity is $\mathcal{O}(12ND^2 + 2N^2D)$ [160]. Stacking L layers leads to a total complexity of $\mathcal{O}(L(12ND^2 + 2N^2D))$. N grows quadratically with frame size and often exceeds D by a large margin, dominating the complexity.

Selective execution saves computation by directly reducing N. In the token reuse setting, only the N' selected tokens are processed through all L encoder layers, while the remaining $N - N'$ tokens are directly reused. The total backbone complexity is $\mathcal{O}(L(12N'D^2 + 2N'^2D))$. This leads to significant savings when $N' \ll N$, as no computations is performed on unselected tokens. In the early exit setting, all N tokens are processed by the first $L' < L$ layers, after which only the selected N' tokens are further processed by the remaining $L - L'$ layers. The overall complexity becomes $\mathcal{O}(L'(12ND^2+2N^2D))+\mathcal{O}((L-L')(12N'D^2+2N'^2D))$, which also benefits from small N', though the initial L' layers still incur a fixed cost over all tokens.

To validate the computational benefits of selective execution, we measure the actual encoder latency across different block execution ratios, as shown in Figure 5.2. Both strategies exhibit a clear nonlinear latency trend: latency decreases sharply as fewer blocks are executed, with diminishing returns at lower execution ratios. Specifically, when only 10% of the blocks are processed, latency decreases from 574.2 ms to 17.8 ms with token reuse, and from 574.6 ms to 155.5 ms with early exit.

These empirical results are consistent with the complexity analysis: both strategies achieve significant speedups at low execution ratios, confirming the effectiveness of



(a) Token reuse (b) Early exit

Figure 5.2: Relationship between number of executed blocks and encoder latency. Input size: 1024×2048, patch size: 16×16.

selective execution in reducing computational cost.

5.2.3 Decoupled Inference and Joint Training

Prior works [94–96] identify informative regions using heuristics, typically by expanding regions around previously detected bounding boxes (bboxes) to account for object motion. For example, Arena [96] expands a fixed number of patches in all directions, while EHCI [95] and FDDIA [94] scale the width and height by fixed factors. Such heuristics often result in redundant selections and thus unnecessary processing overhead. Other methods [19, 152] employ separate policy networks trained on detection results from consecutive frames, where blocks with large motion are more likely to be selected. Without joint optimization with the policy, the detector is not adapted to diverse block-wise inputs, suffering from block artifacts that can degrade accuracy. SViT [101] integrates gating modules into the ViT blocks of the detector to identify informative tokens layer by layer. While effective, it requires full-frame token processing before selection, making it unsuitable for distributed end-edge deployments where lightweight, upfront decision-making is needed to reduce the transmission of

Table 5.2: High-level comparison of SEED and other methods.

Method	Learnable	Decoupled Inference	Joint Training
SEED (ours)	✓	✓	✓
SViT [101]	✓	✗	✓
Arena [96]	✗	✓	✗
BlockCopy [19]	✓	✓	✗

non-IBs. In contrast, SEED introduces a lightweight and learnable DecisionNet that is decoupled from, but jointly trained with, the detector. This design enables accurate and early selection of IBs on resource-constrained end devices. A high-level comparison of SEED and the most relevant methods is shown in Table 5.2.

5.3 SEED Design

In this section, we present the design of SEED, covering the overall framework, the decision network, the block-wise detector, and the three-stage training strategy.

5.3.1 Overview

The overview of SEED is illustrated in Fig. 5.3. The input is first ① down-sampled and passed to the DecisionNet. The ② DecisionNet (Section 5.3.2) generates a decision grid, based on which the ③ BlockDet (Section 5.3.3) performs selective execution. The detection results are then ④ post-processed (e.g., filtering low-confidence bboxes) to produce the final output.

SEED supports two variants, SEED-TR and SEED-EE, each realized by adaptively executing selected blocks within the BlockDet. In SEED-TR, only IBs are

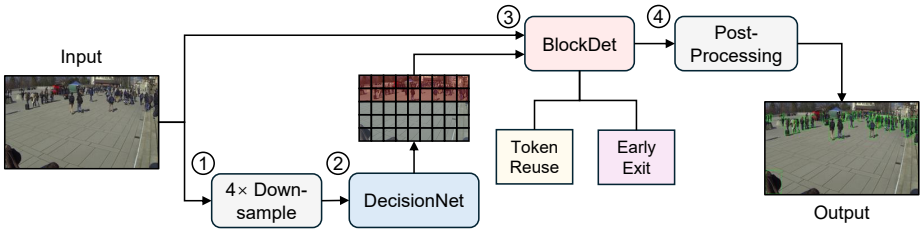


Figure 5.3: Overview of SEED.

fully processed, while non-IBs are skipped by reusing their tokens from previous executions. In SEED-EE, all blocks are processed, but non-IBs exit early after fewer encoder layers. Both DecisionNet and BlockDet are jointly trained to optimize detection performance (Section 5.3.4). For clarity, we refer to the modules in SEED-TR and SEED-EE as DecisionNet-TR/EE and BlockDet-TR/EE, respectively.

5.3.2 Decision Network

The architecture of the DecisionNet is depicted in Figure 5.4. Given the input $\mathcal{I}^k \in \mathbb{R}^{\hat{H}\times\hat{W}\times\hat{C}}$, where $k \geq 1$ denotes the index of the frame in a video sequence, the DecisionNet outputs a decision grid $\mathcal{G}^k$ that guides the subsequent selective execution process.

The input varies across SEED variants. In SEED-TR, which involves token reuse across frames, historical context is critical for guiding block decisions. Accordingly, DecisionNet-TR takes four inputs: 1) the down-sampled current frame $\hat{\mathbf{x}}^k$, 2) the down-sampled previous frame $\hat{\mathbf{x}}^{k-1}$, 3) the previous detection results $\mathcal{D}^{k-1}$, and 4) the previous block decision grid $\mathcal{G}^{k-1}$. In SEED-EE, DecisionNet-EE does not rely on temporal information and only uses $\hat{\mathbf{x}}^k$ as input.

The network architecture is identical for both variants. The input $\mathcal{I}^k$ passes

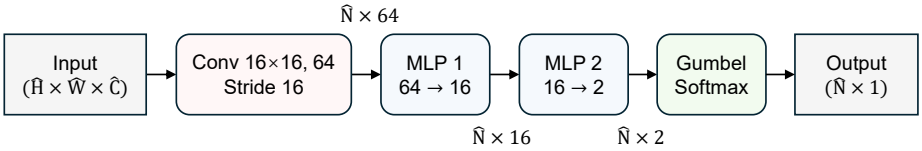


Figure 5.4: Architecture of DecisionNet.

through one Conv layer and two MLP layers to output the logits $\mathcal{P}^k$:

$$\mathcal{P}^k = \mathrm{MLP}_{1,2}(\mathrm{Conv}(\mathcal{I}^k)) \in \mathbb{R}^{\hat{N} \times 2}. \tag{5.3.1}$$

A Gumbel-Softmax layer is then applied to $\mathcal{P}^k$ to produce the decision grid:

$$\mathcal{G}^k = \mathrm{GumbelSoftmax}(\mathcal{P}^k) \in \{0, 1\}^{\hat{N} \times 1}. \tag{5.3.2}$$

5.3.3 Block-Wise Detector

The ViT detector from [96] is adopted as BlockDet. In SEED-TR, it follows the original design (BlockDet-TR), while in SEED-EE, it is extended with an early-exit mechanism (BlockDet-EE). These two variants showcase SEED's extensibility to different selective execution strategies within a unified framework. The architectures of BlockDet-TR and BlockDet-EE are shown in Figure 5.5.

BlockDet-TR: SEED-TR has two phases: full inference and selective inference. For the first frame $\mathbf{x}^k$ $(k = 1)$, BlockDet-TR performs full inference. The frame $\mathbf{x}^1 \in \mathbb{R}^{H \times W \times C}$ is first divided into N non-overlapping patches of size $P \times P$:

$$\mathbf{x}_p^1 = [\mathbf{x}_{p,1}^1;\ \mathbf{x}_{p,2}^1;\ \cdots;\ \mathbf{x}_{p,N}^1], \quad \mathbf{x}_{p,i}^1 \in \mathbb{R}^{P^2 C}. \tag{5.3.3}$$

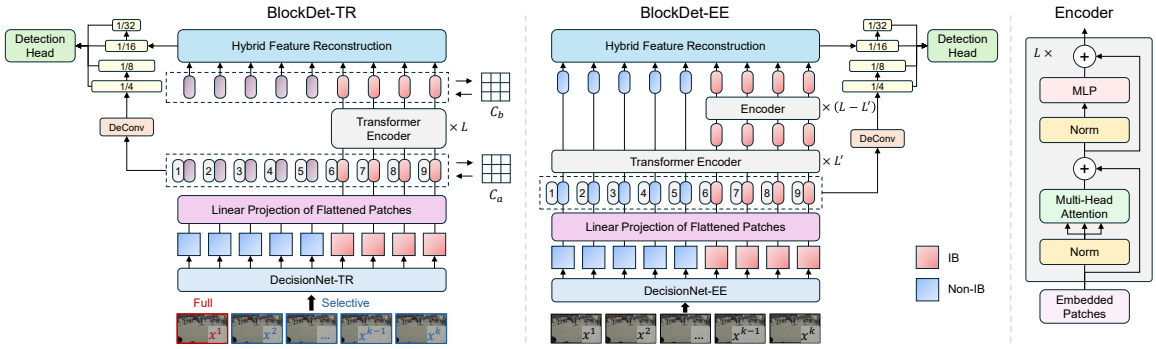


Figure 5.5: Architectures of BlockDet-TR and BlockDet-EE.

Next, the initial tokens $\tilde{\mathbf{z}}_0^1$ are obtained after applying the linear projection $\mathbf{E} \in \mathbb{R}^{P^2C \times D}$:

$$\mathcal{C}_a \leftarrow \tilde{\mathbf{z}}_0^1 = [\mathbf{x}_{p,1}^1\mathbf{E};\ \mathbf{x}_{p,2}^1\mathbf{E};\ \cdots;\ \mathbf{x}_{p,N}^1\mathbf{E}], \tag{5.3.4}$$

where $\tilde{\mathbf{z}}_0^1$ is cached in $\mathcal{C}_a$. Position embeddings $\mathbf{E}_{\text{pos}} \in \mathbb{R}^{N \times D}$ are then added to the initial tokens to preserve positional information as:

$$\mathbf{z}_0^1 = \tilde{\mathbf{z}}_0^1 + \mathbf{E}_{\text{pos}}. \tag{5.3.5}$$

The resulting tokens $\mathbf{z}_0^1$ are processed through L transformer encoder layers [160] to produce the final output $\mathbf{z}_L^1$, which is cached in $\mathcal{C}_b$:

$$\mathbf{z}_\ell^1 = \text{Encoder}(\mathbf{z}_{\ell-1}^1), \qquad \ell = 1, \ldots, L, \tag{5.3.6}$$

$$\mathcal{C}_b \leftarrow \mathbf{z}_L^1. \tag{5.3.7}$$

Each encoder layer consists of a multi-head self-attention (MSA) and a feed-forward

network (MLP), both with residual connections and layer normalization (LN) [160].

$$\tilde{\mathbf{z}}_\ell^1 = \mathrm{MSA}(\mathrm{LN}(\mathbf{z}_{\ell-1}^1)) + \mathbf{z}_{\ell-1}^1, \qquad \ell = 1, \ldots, L, \tag{5.3.8}$$

$$\mathbf{z}_\ell^1 = \mathrm{MLP}(\mathrm{LN}(\tilde{\mathbf{z}}_\ell^1)) + \tilde{\mathbf{z}}_\ell^1, \qquad \ell = 1, \ldots, L. \tag{5.3.9}$$

Notably, the caches $\mathcal{C}_a$ and $\mathcal{C}_b$ are updated in every inference and re-initialized at the next full inference phase. The tokens $\mathbf{z}_L^1$ output from the encoder are then fed to the hybrid feature reconstruction (HFR) layer, which is a single-layer transformer decoder [96, 163] to construct the complete token sequence. To enhance detection performance, a feature pyramid network (FPN) is adopted to extract multi-scale features $\{f_1, f_2, f_3, f_4\}$ from different depths:

$$f_1 = \mathrm{DeConv}_1(\tilde{\mathbf{z}}_0^1), \qquad f_2 = \mathrm{DeConv}_2(\tilde{\mathbf{z}}_0^1), \tag{5.3.10}$$

$$f_3 = \mathrm{HFR}(\mathbf{z}_L^1), \qquad f_4 = \mathrm{Conv}(f_3). \tag{5.3.11}$$

These features are combined and then passed to a detection head [50, 96] to generate detection results (e.g., bboxes, classes, etc.).

For the remaining frames $\mathbf{x}^k$ ($k > 1$), BlockDet-TR performs selective inference. DecisionNet-TR takes as input $\mathcal{I}^k = \{\hat{\mathbf{x}}^k, \hat{\mathbf{x}}^{k-1}, \mathcal{D}^{k-1}, \mathcal{G}^{k-1}\}$ and outputs a block decision grid $\mathcal{G}^k$, which determines the IB indices $\mathcal{B}^k$. Here, $|\mathcal{B}^k| = N'$, where $N' \ll N$. The corresponding tokens $\tilde{\mathbf{z}}_{0,\mathcal{B}^k}^k$ are then extracted for further processing, with position embeddings added based on their original locations in $\mathbf{x}^k$:

$$\mathbf{z}_{0,\mathcal{B}^k}^k = \tilde{\mathbf{z}}_{0,\mathcal{B}^k}^k + \mathbf{E'}_{\mathrm{pos},\mathcal{B}^k}, \quad \mathbf{E'}_{\mathrm{pos},\mathcal{B}^k} \in \mathbb{R}^{N' \times D}. \tag{5.3.12}$$

The resulting tokens $\mathbf{z}^k_{0,\mathcal{B}^k}$ are processed by the encoder to obtain $\mathbf{z}^k_{L,\mathcal{B}^k}$, following Eq. 5.3.6. The newly computed tokens $\mathbf{z}^k_{L,\mathcal{B}^k}$ and the reused tokens $\mathbf{z}^k_{L,\mathcal{B}^k_c}$ retrieved from the cache $\mathcal{C}_b$, are then jointly fed into the HFR layer. Here, $\mathcal{B}^k_c$ denotes the complement of $\mathcal{B}^k$, corresponding to the non-IB indices. The HFR layer recovers the full token sequence and reconstructs spatial and semantic relationships between new and reused tokens, producing a coherent global representation for downstream prediction. Similarly, the input token sequence $\tilde{\mathbf{z}}^k_0$ is formed by merging $\tilde{\mathbf{z}}^k_{0,\mathcal{B}^k}$ with $\tilde{\mathbf{z}}^k_{0,\mathcal{B}^k_c}$, where the latter is retrieved from the cache $\mathcal{C}_a$. After obtaining $\tilde{\mathbf{z}}^k_0$ and $\mathbf{z}^k_L$, the caches are updated accordingly for future use:

$$\mathcal{C}_a \leftarrow \tilde{\mathbf{z}}^k_0, \quad \mathcal{C}_b \leftarrow \mathbf{z}^k_L. \tag{5.3.13}$$

Finally, the multi-scale feature pyramid $\{f_1, f_2, f_3, f_4\}$ is computed, following Eq. 5.3.10 and Eq. 5.3.11. The remaining steps are the same as those in the full inference phase.

BlockDet-EE: For each frame, SEED-EE performs selective inference via early exiting. DecisionNet-EE processes $\hat{\mathbf{x}}^k$ and outputs a decision grid $\mathcal{G}^k$ that indicates whether each block should undergo full or shallow processing by BlockDet-EE.

The input patches $\mathbf{x}^k_p$ are first transformed into tokens $\mathbf{z}^k_0 \in \mathbb{R}^{N\times D}$ following Eq. 5.3.3– 5.3.5. These tokens are then processed by the first L' layers of the encoder:

$$\mathbf{z}^k_\ell = \text{Encoder}(\mathbf{z}^k_{\ell-1}), \quad \ell = 1, \ldots, L'. \tag{5.3.14}$$

At this point, the early exit mechanism is triggered. For non-IBs $\mathcal{B}^k_c$, inference halts and their intermediate features $\mathbf{z}^k_{L',\mathcal{B}^k_c}$ are directly retained. For IBs $\mathcal{B}^k$, computation

continues through the remaining layers:

$$\mathbf{z}^k_{\ell,\mathcal{B}^k} = \text{Encoder}(\mathbf{z}^k_{\ell-1,\mathcal{B}^k}), \quad \ell = L'+1, \ldots, L. \tag{5.3.15}$$

The final token sequence $\mathbf{z}^k_L$ is reconstructed within the HFR layer, which first merges the early-exited tokens $\mathbf{z}^k_{L',\mathcal{B}^k_c}$ and the fully-processed tokens $\mathbf{z}^k_{L,\mathcal{B}^k}$, and then refines the combined sequence to recover cross-token relationships. Subsequent processing, including multi-scale feature fusion and detection head inference, follows the same procedure as BlockDet-TR.

Note that the early exit point L' in BlockDet-EE is tunable and can be adjusted according to application-specific requirements and the target accuracy-latency trade-off. For simplicity and to showcase the feasibility of BlockDet-EE, we set $L' = L/4$ throughout this study.

5.3.4 Joint Training

Naively training the DecisionNet and BlockDet jointly from scratch using only task loss is problematic. At the early stage of training, the DecisionNet produces unstable block decisions, failing to provide meaningful spatial guidance for the BlockDet. Meanwhile, the BlockDet itself lacks basic object detection capability due to random initialization. These two issues reinforce each other negatively: inaccurate block selections weaken the training signals for the BlockDet, while the inadequately trained BlockDet provides ineffective feedback for improving the DecisionNet. This mutual dependence creates a cold-start problem that hinders convergence and degrades overall detection performance.

To address this issue, we propose a three-stage training strategy:

BlockDet Pre-training: The BlockDet is first initialized with a backbone pre-trained on a large-scale dataset [157]. To adapt it to the selective execution setting, we further fine-tune it using partially masked inputs by randomly dropping 50% of the blocks during training to simulate sparse spatial patterns. This strategy improves the robustness of the BlockDet to partial inputs and prepares it for the subsequent block-wise selective inference.

DecisionNet Warm-Up: We then train the DecisionNet independently using supervision from a pseudo-label decision grid $\mathcal{G}$. This grid can be derived from ground-truth annotations. The training objective is to make the predicted grid $\hat{\mathcal{G}}$ approximate $\mathcal{G}$, providing a more informative initialization. The grid loss is defined in Eq. 5.3.16:

Joint Optimization: Once both networks are warmed up, we jointly train the DecisionNet and BlockDet in an end-to-end manner:

$$\mathcal{L}_{\text{grid}} = \mathcal{L}_{\text{BCE}}(\hat{\mathcal{G}}, \mathcal{G}), \tag{5.3.16}$$

$$\mathcal{L}_{\text{complexity}} = \mathcal{L}_{\text{MSE}}(\tau,\ \frac{1}{\hat{N}} \sum_{i=1}^{\hat{N}} \hat{\mathcal{G}}_i), \tag{5.3.17}$$

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{task}} + \alpha \cdot \mathcal{L}_{\text{grid}} + \beta \cdot \mathcal{L}_{\text{complexity}}. \tag{5.3.18}$$

Here, $\mathcal{L}_{\text{task}}$ refers to the standard detection loss from BlockDet (e.g., classification and bbox regression) [50], while $\mathcal{L}_{\text{grid}}$ encourages block selection to stay close to meaningful spatial regions [164]. Additionally, $\mathcal{L}_{\text{complexity}}$ constrains the overall execution cost to align with a target block selection ratio τ [100, 101]. The balancing weights α and β control the contribution of each auxiliary loss term. This staged strategy enables stable convergence and allows the DecisionNet to learn a selection policy that is both effective and computationally efficient.

Pseudo-Label Grid: To supervise the training of the DecisionNet, we construct a pseudo-label grid $\mathcal{G}$ that serves as an approximate ground truth for block importance. For SEED-TR, $\mathcal{G}$ is generated by projecting ground-truth bboxes onto the block grid, where each block is marked as 1 if it overlaps with any object, and 0 otherwise. This provides a coarse but effective approximation of the blocks that require full processing. For SEED-EE, the construction of $\mathcal{G}$ additionally considers whether a block can be handled by shallow inference. We first mark all blocks overlapping with ground-truth objects as 1. Then, a shallow detector (with depth L') is used to perform inference on the training set. If a ground-truth object is successfully detected by the shallow detector, the corresponding block is re-labeled as 0. The rationale is that blocks correctly processed by early layers do not need deeper computation, and can therefore be processed with early exit.

5.4 Evaluation

In this section, we evaluate the performance of SEED through extensive experiments. We begin with the experimental setup, followed by comparisons with several baselines on two benchmark datasets and a real-world testbed.

5.4.1 Experimental Setup

The hardware platform consists of the NVIDIA Jetson TX2 and the Dell desktop introduced in Section 2.5. The two devices are connected via a D-Link AX4800 router over a 2.4GHz Wi-Fi network, with an average bandwidth of 20.1 Mbps measured using `iperf`. Evaluation is conducted in two settings: the benchmark evaluation

(Section 5.4.5) runs all computation on the desktop for controlled comparison, while the testbed evaluation (Section 5.4.6) distributes computation between the Jetson TX2 (as a smart camera) and the desktop (as an edge server) to simulate a real-world edge video analytics deployment.

5.4.2 Datasets and Metrics

We evaluate SEED on two representative pedestrian detection datasets mentioned in Section 2.3: MOT17 and WildTrack. Each video sequence is split into 75% for training the DecisionNet and BlockDet, and 25% for testing. All frames are resized to 1024×2048 for consistency. For evaluation, we use mAP@0.5 to measure accuracy, and report end-to-end latency and network traffic to measure efficiency, as introduced in Section 2.4.

5.4.3 SEED Configuration

The DecisionNet has $\sim$0.1M parameters, less than 0.3% of BlockDet's 44.6M. The BlockDet adopts Faster-RCNN [50], a two-stage detector with ViT-Small as the backbone. The network configuration follows [96], with encoder depth $L = 12$, embedding dimension $D = 384$, and patch size $= 16 \times 16$. One block in DecisionNet corresponds to a 4×4 grid of patches. The pre-trained weights of ViT-Small are from [157]. The target block selection ratio $\tau = 20\%$, and the balancing hyperparameters $\alpha = 4, \beta = 5$. The BlockDet is first pre-trained for 60 epochs with an AdamW optimizer [165] (learning rate 1e-4, weight decay 1e-3). The DecisionNet is then warmed up for 30 epochs with AdamW (learning rate 1e-3, weight decay 1e-4). The joint training is finally conducted for 50 epochs using the same settings as in the BlockDet pre-training.

5.4.4 Baselines

SEED is compared against the following baselines. To ensure a fair comparison, all methods use the same detector framework, while their training strategies and block selection mechanisms follow their respective original designs.

- **Full-Frame Detector (FD):** This is a standard baseline where the detector (with L encoder layers) performs full-frame inference without any form of selective execution.
- **FD-Quarter:** A lightweight variant of FD in which all frames are processed by a shallower detector with $L' = L/4$ encoder layers.
- **FD-Random:** A naive baseline where 30% of the blocks are randomly selected for processing, serving as a reference to assess the benefit of content-aware selection.
- **BlockCopy [19]:** A separate policy network identifies IBs, which are then processed by the detector. Features corresponding to non-IBs are cached and reused across frames to reduce redundant computation.
- **SViT [101]:** Layer-wise token pruning is applied to each frame independently. The gating module inside each ViT block selects informative tokens for further processing, while unselected tokens reuse representations from the previous layer.
- **Arena-TR [96]:** A full frame is processed periodically to obtain a complete set of tokens, which are then cached. For subsequent frames, IBs are selected based on previous detection results. Only tokens from the selected blocks are

updated, while those from non-IBs are reused to reconstruct a complete token sequence for detection.

- **Arena-EE:** A variant of Arena adapted for early exit. All tokens pass through the first $L' = L/4$ layers of the encoder. Only tokens corresponding to IBs continue through the remaining layers, while others exit early without further computation.

.

5.4.5 Benchmark Evaluation

We evaluate SEED against the selected baselines on MOT17 and WildTrack under token reuse and early exit. For fair comparison, SEED-TR is compared with FD, BlockCopy, SViT and Arena-TR, while SEED-EE is compared with FD, FD-Quarter, FD-Random and Arena-EE. The results are reported in Table 5.3 and Table 5.4.

In the token reuse setting, SEED-TR gains 82.0% mAP on MOT17, with only a 1.6% drop from FD, while executing only 27.2% of the blocks and reducing latency by 75.5%. In contrast, BlockCopy, lacking joint optimization, suffers from block artifacts and shows a larger accuracy drop (4.6%) despite executing more blocks (37.8%). SViT reaches the same accuracy as SEED-TR (82.0%) with slightly higher block usage (30.1%) and latency. Arena-TR determines IBs using motion heuristics, executing more blocks (41.6%) and reducing latency (66.5%) less than SEED-TR. Similar trends are observed on WildTrack. SEED-TR reaches 76.1% mAP with only 28.5% of blocks executed and a 74.4% latency reduction. BlockCopy and Arena-TR require significantly more computation (up to 45.4%) yet show lower accuracy. SViT

Table 5.3: Benchmark evaluation on two datasets for token reuse.

Dataset	Method	Accuracy (%,%)	Blocks (%)	Latency (ms)
MOT17	SEED-TR	82.0 (↓ 1.6)	**27.2**	**168.6** (↓ 75.5%)
	FD	**83.6**	100	689.1
	BlockCopy	79.0 (↓ 4.6)	37.8	218.7 (↓ 68.3%)
	SViT	82.0 (↓ 1.6)	30.1	194.2 (↓ 71.8%)
	Arena-TR	81.4 (↓ 2.2)	41.6	230.8 (↓ 66.5%)
WildTrack	SEED-TR	76.1 (↓ 1.7)	**28.5**	**176.6** (↓ 74.4%)
	FD	**77.8**	100	688.5
	BlockCopy	72.9 (↓ 4.9)	38.9	222.8 (↓ 67.6%)
	SViT	76.2 (↓ 1.6)	31.4	198.7 (↓ 71.1%)
	Arena-TR	75.7 (↓ 2.1)	45.4	248.7 (↓ 63.9%)

remains competitive in accuracy (76.2%) but executes more blocks (31.4%) and is overall less efficient than SEED-TR.

In the early exit setting, SEED-EE achieves 83.3% mAP on MOT17, closely matching FD (83.6%) while executing only 22.6% of blocks and reducing latency by 57.4%. FD-Quarter, although faster due to its shallow backbone, suffers from a significant accuracy drop of 11.4%. FD-Random reaches similar latency as SEED-EE but incurs an 8.4% drop in accuracy due to its naive block selection. Arena-EE achieves 82.8% accuracy with 48.8% latency reduction, but still executes significantly more blocks (43.4%) due to its reliance on past detection results. On WildTrack, the trend remains consistent. SEED-EE continues to deliver better results, achieving 77.2% mAP with 23.7% of blocks executed and 56.9% latency reduction. In contrast, FD-Random and FD-Quarter still show poor accuracy (10.1%–13.4% lower than FD), while Arena-EE, although competitive in accuracy, consumes nearly twice the number of blocks as SEED-EE.

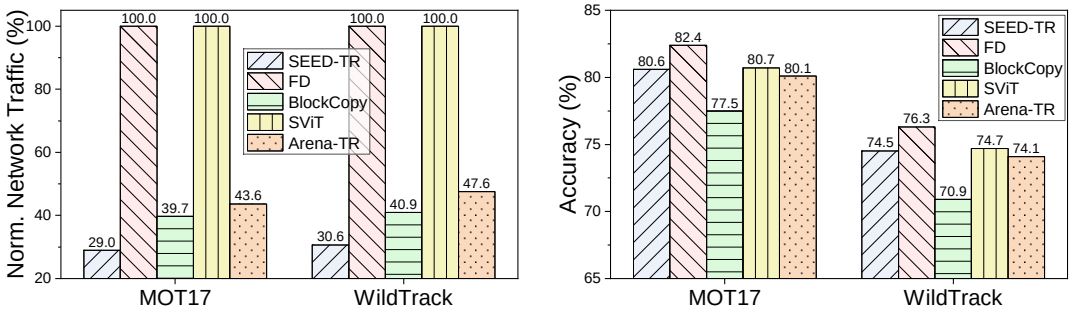
Table 5.4: Benchmark evaluation on two datasets for early exit.

Dataset	Method	Accuracy (%,%)	Blocks (%)	Latency (ms)
MOT17	SEED-EE	83.3 (↓ 0.3)	**22.6**	293.2 (↓ 57.4%)
	FD	**83.6**	100	688.3
	FD-Quarter	72.2 (↓ 11.4)	100	**264.8** (↓ 61.5%)
	FD-Random	75.2 (↓ 8.4)	30.0	311.7 (↓ 54.7%)
	Arena-EE	82.8 (↓ 0.8)	43.4	352.4 (↓ 48.8%)
WildTrack	SEED-EE	77.2 (↓ 0.6)	**23.7**	296.1 (↓ 56.9%)
	FD	**77.8**	100	687.6
	FD-Quarter	64.4 (↓ 13.4)	100	**264.5** (↓ 61.5%)
	FD-Random	67.7 (↓ 10.1)	30.0	312.2 (↓ 54.6%)
	Arena-EE	76.8 (↓ 1.0)	46.8	365.4 (↓ 46.9%)

5.4.6 Testbed Evaluation

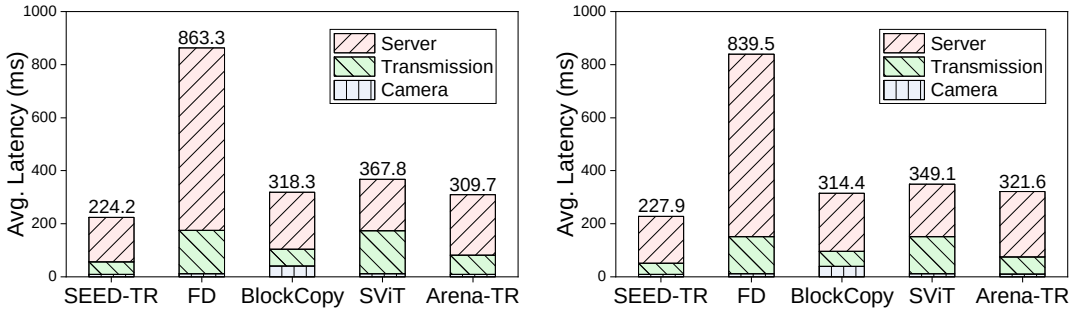
As described in Section 5.4.1, our testbed consists of a camera and an edge server. To minimize communication overhead, we adopt different optimizations for token reuse and early exit. In both settings, IBs are identified on the camera, merged into a compact frame, compressed, and transmitted to the server. For early exit, a downsampled copy of the original frame is also sent; the server then upsamples it to the original size and replaces the corresponding blocks with the received IBs to reconstruct the full frame for downstream processing. For fairness, these optimizations are applied to all the baselines as well. Figures 5.6 and 5.8 show the normalized network traffic and accuracy of different methods in both settings across two datasets. Figures 5.7 and 5.9 further break down the average end-to-end latency into camera time, transmission time, and server time. Due to the slight information loss during compression, the accuracy across all methods shows a minor decrease of less than 2%, with their relative performance consistent with the benchmark evaluation in Tables 5.3–5.4.

As depicted in Figures 5.7 and 5.9, server-side inference dominates end-to-end



(a) Normalized network traffic (b) Accuracy

Figure 5.6: (a) Normalized network traffic and (b) accuracy of different methods in the token reuse setting on two datasets.

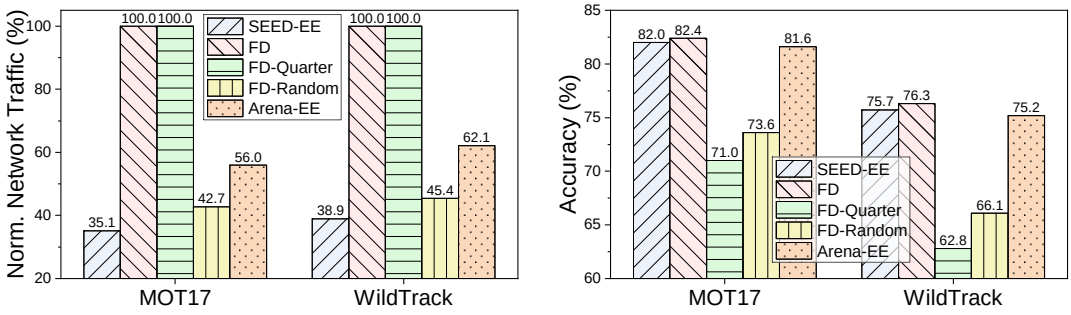


(a) MOT17 (b) WildTrack

Figure 5.7: Average end-to-end latency of different methods in the token reuse setting on two datasets.

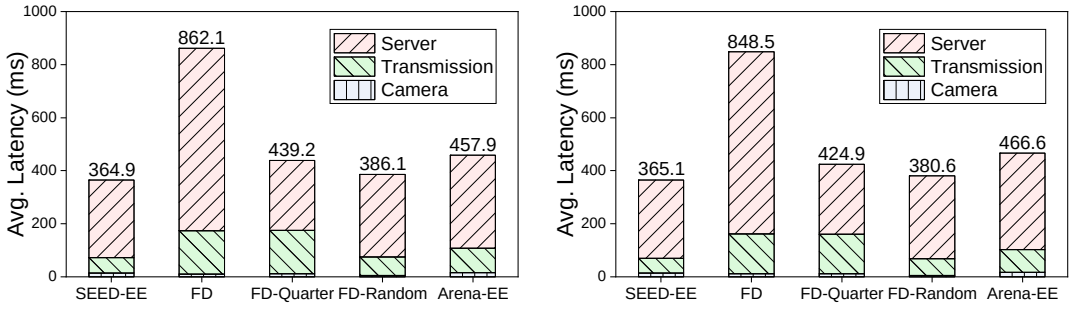
latency across all methods due to the high computational cost of running the detector. FD, which transmits and processes full frames without selective execution, incurs the highest network traffic and latency. SEED-TR achieves the lowest latency, demonstrating reductions of 74.0% and 72.9% on MOT17 and WildTrack, respectively, compared to FD. SViT, with its gating module embedded within the detector, requires transmitting full frames and thus offers limited communication savings. In addition, BlockCopy and Arena-TR execute more blocks than SEED-TR, leading to higher communication and computation overhead.

In the early exit scenario, SEED-EE achieves accuracy comparable to FD while



(a) Normalized network traffic (b) Accuracy

Figure 5.8: (a) Normalized network traffic and (b) accuracy of different methods in the early exit setting on two datasets.



(a) MOT17 (b) WildTrack

Figure 5.9: Average end-to-end latency of different methods in the early exit setting on two datasets.

reducing latency by 57.7% and 57.0% on MOT17 and WildTrack, respectively. Notably, despite being around 15% faster than SEED-EE in server-side processing, FD-Quarter transmits full frames, which increases communication time and diminishes its advantage, resulting in even longer end-to-end latency. FD-Random achieves similar latency to SEED-EE but suffers significant accuracy degradation due to its random selection strategy. Arena-EE, while offering slightly lower accuracy than SEED-EE, processes a greater number of blocks, leading to higher transmission and server times.

5.4.7 Visualization

To further illustrate the effectiveness of SEED, Figure 5.10 presents a visualization of multiple representative scenes for both SEED-TR and SEED-EE, along with the proportion of selected IBs displayed below each frame. Selected IBs are highlighted in red and detection results are marked as green bboxes. It shows that the DecisionNet makes accurate and content-aware block decisions. In SEED-TR, IBs tend to cluster in regions with high object density or severe occlusion, while non-IBs are mostly associated with sparse areas or slowly moving objects. A similar pattern is observed in SEED-EE, where IBs concentrate around challenging regions, i.e., those with dense objects, and non-IBs are typically found in easier areas with few or isolated objects. These patterns are consistent with the intended behavior of each variant.

5.5 Conclusion

In this chapter, we presented SEED, an end-to-end trainable framework for selective execution in ViT-based object detection. SEED leverages a lightweight and content-aware DecisionNet to identify informative blocks, enabling the downstream BlockDet to reduce computation through either token reuse (SEED-TR) or early exit (SEED-EE) strategies. Both networks are trained jointly to achieve optimal block selection and execution. Extensive evaluations on public datasets and a real-world testbed demonstrate that SEED accelerates edge video analytics by reducing computation and communication costs, with only minimal accuracy loss.

Figure 5.10: Visualization of SEED-TR and SEED-EE across multiple scenes.

Chapter 6

Conclusion

6.1 Summary

This thesis addresses the challenge of accelerating VAPs on resource-constrained edge platforms by exploring adaptive and content-aware strategies that reduce redundant computation and communication. We propose three approaches: FastTuner, BlockHybrid, and SEED, that address different dimensions of the accuracy-efficiency trade-off.

FastTuner introduces a runtime configuration optimization framework for MOT. By learning heatmap representations offline and integrating configuration selection with tracking in a shared model, it efficiently chooses the best resolution-backbone pair for the pipeline, enabling low-overhead and adaptable execution.

BlockHybrid targets efficient object detection through fine-grained block-wise conditional execution. It distinguishes between "hard" and "easy" blocks using a policy network, assigning them to a heavy-weight detector or lightweight tracker, respectively, to reduce redundant computation and communication.

SEED advances this direction by coupling block selection and execution in an end-to-end trainable architecture tailored for ViT-based detection pipelines. A lightweight, context-aware DecisionNet identifies informative regions, enabling the BlockDet to selectively process them via token reuse (SEED-TR) or early exit (SEED-EE), achieving efficient inference without compromising accuracy.

6.2 Limitations

While the proposed frameworks demonstrate superior performance, several limitations remain.

In FastTuner, the optimal configuration is decided every K frames under the assumption that video content remains relatively stable over short intervals. However, this assumption may not hold in highly dynamic scenes. Future work can explore adaptive strategies for determining K based on scene variation, enabling the framework to respond more effectively to rapid changes. Moreover, the current design considers only two control knobs: frame resolution and backbone model. Incorporating additional knobs such as frame rate and quantization parameter (QP) could enhance adaptability and generality. As the configuration space expands, however, the number of heatmaps and associated computation also increases. Efficient sampling or approximation techniques should be investigated to mitigate this overhead and preserve runtime efficiency.

A major limitation of BlockHybrid lies in its decoupled architecture. Block selection and execution are performed separately, and the block-wise detector and lightweight tracker operate independently. This prevents end-to-end training of the full pipeline. While block-wise fine-tuning can partially alleviate block artifacts, a mismatch often exists between offline and online stages. During offline fine-tuning, reused features typically come from adjacent frames with minimal temporal drift, whereas in online stages, cached features used for block merging may originate from distant frames, introducing severe inconsistencies. These issues necessitate periodic full-frame inference to refresh cached features and tracker references. As shown in Table 4.7, increasing the update interval amplifies feature inconsistencies and leads to a sharp drop in accuracy.

SEED addresses this limitation by coupling block selection and execution within

an end-to-end training and inference pipeline. This integration enables joint optimization of both components, effectively reducing block artifacts and improving coordination between them. However, as the DecisionNet is trained offline, its performance can degrade when camera viewpoints or scene conditions change significantly, such as when cameras are repositioned or deployed in new environments. In such scenarios, re-training or fine-tuning may be required to ensure reliable block selection. Moreover, although SEED is currently focused on object detection, the framework is inherently general and can be extended to other vision tasks, such as segmentation, with only minor architectural modifications and task-specific training strategies.

6.3 Future Work

Beyond addressing limitations of individual approaches, we have identified several directions for future research that are critical to the development of real-time and scalable EVA systems.

First, realizing fully adaptive pipelines remains an open challenge. Current frameworks often rely on manually defined intervals or static policies (e.g., fixed K or fixed thresholds) to control adaptation frequency, which may not generalize well across diverse or unpredictable video streams. Future systems must jointly optimize temporal scheduling (e.g., adaptive K for configuration switching), spatial selection (e.g., identifying informative blocks), and model configuration (e.g., resolution, backbone, depth) under strict latency and energy budgets. For instance, consider a traffic monitoring system deployed at a busy urban intersection. During peak hours, rapid scene changes (e.g., vehicles turning, pedestrians crossing) may require high spatial resolution and short adaptation intervals (i.e., small K), with fine-grained block selection

to capture dynamic regions. At the same time, the system must choose a powerful model to maintain detection accuracy. However, this high-cost configuration is unsustainable during low-bandwidth periods or when power is constrained. Instead of adapting each dimension independently, a joint scheduler must reason about the trade-offs, for example, using a lower-resolution input with a stronger model and longer reuse interval, or increasing resolution but reducing the number of selected blocks, to stay within the system's budget while maximizing task performance. Designing such a unified controller that can coordinate these interacting dimensions in response to both content dynamics and runtime constraints is a key direction for future research.

Second, integrating the proposed frameworks into a complete system stack is essential for deployment in real-world edge environments. While this thesis focuses primarily on optimizing inference pipelines, practical deployments involve many additional system-level components, including input buffering, communication scheduling, and multi-camera coordination. In distributed settings, unstable network conditions such as jitter and packet loss, can affect system performance. Delayed or missing frames may lead to outdated inputs for configuration selection, suboptimal block decisions, or corrupted feature reuse, ultimately degrading accuracy and stability. In multi-camera settings, naively applying existing methods by treating each video stream independently leads to complexity that scales linearly with the number of cameras. In practice, overlapping fields of view (FoVs) are common, particularly in dense environments like traffic intersections, where multiple roadside cameras may observe the same or adjacent regions. Therefore, to reduce redundant computation, a system must go beyond isolated decisions and instead coordinate processing across

streams, jointly selecting the most informative viewpoints and regions.

Third, while this thesis focuses on per-frame inference, deeper integration of temporal modeling and memory-aware mechanisms could significantly enhance system efficiency. Human perception naturally accumulates information over time, allowing us to ignore static or predictable regions. Similarly, future systems should leverage long-term spatio-temporal context to suppress redundant processing. For example, a block that has remained visually static across several frames could be skipped entirely, or updated at a lower frequency, while attention is directed to regions with motion or novel activity. This requires not only memory-aware tracking modules, but also inference models that are capable of selectively updating representations based on content novelty. Lightweight memory modules, causal temporal attention, and event-triggered inference policies are promising techniques to explore in this space.

Finally, generalizing selective execution to a broader range of tasks beyond object detection and tracking is an important avenue for future research. While this thesis demonstrates results on object detection and tracking, many practical applications, such as semantic segmentation, action recognition, multi-modal fusion, and scene-level understanding, could also benefit from adaptive and context-aware execution. For example, in retail analytics, selectively analyzing only store zones with customers could reduce processing cost while preserving key behavioral insights. Designing task-agnostic decision modules or training objectives that generalize across different vision tasks would make such frameworks more widely applicable.

Overall, the proposed frameworks offer promising building blocks toward efficient edge video analytics. To fully realize this vision, future research must focus on unified, cross-layer solutions that co-optimize models, decision modules, and runtime

infrastructure. Such efforts are essential to achieving truly responsive, intelligent, and scalable video analytics in real-world edge deployments.

Bibliography

[1] Elly Cosgrove. One billion surveillance cameras will be watching around the world in 2021. https://www.cnbc.com/2019/12/06/one-billion-surveillance-cameras-will-be-watching-globally-in-2021.html, 2022. Accessed: September 9, 2025.

[2] Renjie Xu, Saiedeh Razavi, and Rong Zheng. Edge video analytics: A survey on applications, systems and enabling techniques. *IEEE Commun. Surv. Tutor.*, 25(4):2951–2982, 2023.

[3] Amita Potnis. Managing unstructured data growth requires a fresh approach. https://www.quantum.com/globalassets/documents/idc-vendor-spotlight.pdf, 2024. Accessed: September 9, 2025.

[4] Shibo Wang, Shusen Yang, and Cong Zhao. SurveilEdge: Real-time video query based on collaborative cloud-edge deep learning. In *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, pages 2519–2528, 2020.

[5] Shuyao Shi, Jiahe Cui, Zhehao Jiang, Zhenyu Yan, Guoliang Xing, Jianwei Niu,

and Zhenchao Ouyang. VIPS: Real-time perception fusion for infrastructure-assisted autonomous driving. In *Proc. Annu. Int. Conf. Mobile Comput. Netw. (MobiCom)*, pages 133–146, 2022.

[6] Qingyang Zhang, Hui Sun, Xiaopei Wu, and Hong Zhong. Edge video analytics for public safety: A review. *Proc. IEEE*, 107(8):1675–1696, 2019.

[7] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodik, Leana Golubchik, Minlan Yu, Paramvir Bahl, and Matthai Philipose. VideoEdge: Processing camera streams using hierarchical clusters. In *Proc. IEEE/ACM Symp. Edge Compt. (SEC)*, pages 115–131, 2018.

[8] Ran Xu, Jinkyu Koo, Rakesh Kumar, Peter Bai, Subrata Mitra, Sasa Misailovic, and Saurabh Bagchi. VideoChef: Efficient approximation for streaming video processing pipelines. In *Proc. USENIX Conf. Annu. Tech. Conf. (USENIX ATC)*, page 43–55, 2018.

[9] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J. Freedman. Live video analytics at scale with approximation and delay-tolerance. In *Proc. USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, page 377–392, 2017.

[10] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. Chameleon: scalable adaptation of video analytics. In *Proc. Conf. ACM Special Interest Group Data Comm. (SIGCOMM)*, pages 253–266, 2018.

[11] Mu Yuan, Lan Zhang, Fengxiang He, Xueting Tong, and Xiang-Yang Li. InFi: End-to-end learnable input filter for resource-efficient mobile-centric inference.

In *Proc. Annu. Int. Conf. Mobile Comput. Netw. (MobiCom)*, page 228–241, 2022.

[12] Yuanqi Li, Arthi Padmanabhan, Pengzhan Zhao, Yufei Wang, Guoqing Harry Xu, and Ravi Netravali. Reducto: On-camera filtering for resource-efficient real-time video analytics. In *Proc. Conf. ACM Special Interest Group Data Comm. (SIGCOMM)*, pages 359–376, 2020.

[13] Chengyi Qu, Rounak Singh, Alicia Esquivel-Morel, and Prasad Calyam. Learning-based multi-drone network edge orchestration for video analytics. In *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, pages 1219–1228, 2022.

[14] Kongyange Zhao, Zhi Zhou, Xu Chen, Ruiting Zhou, Xiaoxi Zhang, Shuai Yu, and Di Wu. EdgeAdaptor: Online configuration adaption, model selection and resource provisioning for edge DNN inference serving at scale. *IEEE Trans. Mobile Comput.*, 22(10):5870–5886, 2022.

[15] Ran Xu, Fangzhou Mu, Jayoung Lee, Preeti Mukherjee, Somali Chaterji, Saurabh Bagchi, and Yin Li. SmartAdapt: Multi-branch object detection framework for videos on mobiles. In *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 2528–2538, 2022.

[16] Ran Xu, Rakesh Kumar, Pengcheng Wang, Peter Bai, Ganga Meghanath, Somali Chaterji, Subrata Mitra, and Saurabh Bagchi. ApproxNet: Content and contention-aware video object classification system for embedded clients. *ACM Trans. Sens. Netw. (TOSN)*, 18(1):1–27, 2021.

[17] Fang Dong, Huitian Wang, Dian Shen, Zhaowu Huang, Qiang He, Jinghui

Zhang, Liangsheng Wen, and Tingting Zhang. Multi-exit DNN inference acceleration based on multi-dimensional optimization for edge intelligence. *IEEE Trans. Mobile Comput.*, 22(9):5389–5405, 2022.

[18] Biyi Fang, Xiao Zeng, Faen Zhang, Hui Xu, and Mi Zhang. FlexDNN: Input-adaptive on-device deep learning for efficient mobile vision. In *Proc. IEEE/ACM Symp. Edge Compt. (SEC)*, pages 84–95, 2020.

[19] Thomas Verelst and Tinne Tuytelaars. BlockCopy: High-resolution video processing with block-sparse feature propagation and online policies. In *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, pages 5158–5167, 2021.

[20] Thomas Verelst and Tinne Tuytelaars. SegBlocks: Block-based dynamic resolution networks for real-time segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 45(2):2400–2411, 2022.

[21] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *ACM Comput. Surv. (CSUR)*, 54(10s):1–41, 2022.

[22] Shixing Yu, Tianlong Chen, Jiayi Shen, Huan Yuan, Jianchao Tan, Sen Yang, Ji Liu, and Zhangyang Wang. Unified visual transformer compression. In *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2022.

[23] Zhihang Yuan, Chenhao Xue, Yiqi Chen, Qiang Wu, and Guangyu Sun. Ptq4vit: Post-training quantization for vision transformers with twin uniform quantization. In *Proc. Eur. Conf. Comput. Vis. (ECCV)*, pages 191–207, 2022.

[24] Sachin Mehta and Mohammad Rastegari. Separable self-attention for mobile vision transformers. *arXiv preprint arXiv:2206.02680*, 2022.

[25] Abdelrahman Shaker, Muhammad Maaz, Hanoona Rasheed, Salman Khan, Ming-Hsuan Yang, and Fahad Shahbaz Khan. SwiftFormer: Efficient additive attention for transformer-based real-time mobile vision applications. In *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, pages 17425–17436, 2023.

[26] Renjie Xu, Keivan Nalaie, and Rong Zheng. BlockHybrid: Accelerating object detection pipelines with hybrid block-wise execution. *IEEE Internet Things J.*, 12(13):24148–24158, 2025.

[27] Yiding Wang, Weiyan Wang, Junxue Zhang, Junchen Jiang, and Kai Chen. Bridging the edge-cloud barrier for real-time advanced vision analytics. In *Proc. USENIX Conf. Hot Topics Cloud Comput.*, page 18, 2019.

[28] Yiding Wang, Weiyan Wang, Duowen Liu, Xin Jin, Junchen Jiang, and Kai Chen. Enabling edge-cloud video analytics for robotics applications. *IEEE Trans. Cloud Comput.*, 11(2):1500–1513, 2022.

[29] Kuntai Du, Ahsan Pervaiz, Xin Yuan, Aakanksha Chowdhery, Qizheng Zhang, Henry Hoffmann, and Junchen Jiang. Server-driven video streaming for deep learning inference. In *Proc. Conf. ACM Special Interest Group Data Comm. (SIGCOMM)*, pages 557–570, 2020.

[30] Huaizheng Zhang, Meng Shen, Yizheng Huang, Yonggang Wen, Yong Luo, Guanyu Gao, and Kyle Guan. A serverless cloud-fog platform for DNN-based

video analytics with incremental learning. *arXiv preprint arXiv:2102.03012*, 2021.

[31] Luyang Liu, Hongyu Li, and Marco Gruteser. Edge assisted real-time object detection for mobile augmented reality. In *Proc. Annu. Int. Conf. Mobile Comput. Netw. (MobiCom)*, pages 1–16, 2019.

[32] Bin Qian, Zhenyu Wen, Junqi Tang, Ye Yuan, Albert Y Zomaya, and Rajiv Ranjan. OsmoticGate: Adaptive edge-based real-time video analytics for the internet of things. *IEEE Trans. Comput.*, 72(4):1178–1193, 2022.

[33] Shanzhi Chen, Jinling Hu, Yan Shi, Li Zhao, and Wen Li. A vision of C-V2X: Technologies, field testing, and challenges with chinese development. *IEEE Internet Things J.*, 7(5):3872–3881, 2020.

[34] Yuqi Dong, Guanyu Gao, Ran Wang, and Zhisheng Yan. Collaborative video analytics on distributed edges with multiagent deep reinforcement learning. *arXiv preprint arXiv:2211.03102*, 2022.

[35] Hongpeng Guo, Beitong Tian, Zhe Yang, Bo Chen, Qian Zhou, Shengzhong Liu, Klara Nahrstedt, and Claudiu Danilov. DeepStream: Bandwidth efficient multi-camera video streaming for deep learning analytics. *arXiv preprint arXiv:2306.15129*, 2023.

[36] Liming Ge, Wei Bao, Dong Yuan, and Bing B Zhou. Edge-assisted deep video denoising and super-resolution for real-time surveillance at night. In *Proc. Annu. Int. Conf. Mobile Comput. Netw. (MobiCom)*, pages 783–785, 2022.

[37] Hui Sun, Qiyuan Li, Kewei Sha, and Ying Yu. ElasticEdge: An intelligent elastic edge framework for live video analytics. *IEEE Internet Things J.*, 9(22):23031–23046, 2022.

[38] Syed Waqas Zamir, Aditya Arora, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, Ming-Hsuan Yang, and Ling Shao. Multi-stage progressive image restoration. In *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 14821–14831, 2021.

[39] Xu Qin, Zhilin Wang, Yuanchao Bai, Xiaodong Xie, and Huizhu Jia. FFA-Net: Feature fusion attention network for single image dehazing. In *Proc. AAAI Conf. Artif. Intell. (AAAI)*, volume 34, pages 11908–11915, 2020.

[40] Gary Bradski. The OpenCV library. *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, 25(11):120–123, 2000.

[41] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, volume 1, pages 886–893, 2005.

[42] David G Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vis.*, 60:91–110, 2004.

[43] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. IEEE/CVF Conf. Comput. Vision and Pattern Recognit. (CVPR)*, pages 770–778, 2016.

[44] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets:

Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[45] Andrews Sobral and Antoine Vacavant. A comprehensive review of background subtraction algorithms evaluated with synthetic and real videos. *Comput. Vis. Image Underst.*, 122:4–21, 2014.

[46] Sandeep Singh Sengar and Susanta Mukhopadhyay. A novel method for moving object detection based on block based frame differencing. In *Proc. Int. Conf. Recent Adv. Inf. Technol. (RAIT)*, pages 467–472, 2016.

[47] Anshuman Agarwal, Shivam Gupta, and Dushyant Kumar Singh. Review of optical flow technique for moving object detection. In *Proc. Int. Conf. Contemp. Comput. Inform. (IC3I)*, pages 409–413, 2016.

[48] Sepehr Aslani and Homayoun Mahdavi-Nasab. Optical flow based moving object detection and tracking for traffic surveillance. *Int. J. Electr. Comput. Energ. Electron. Commun. Eng.*, 7(9):1252–1256, 2013.

[49] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. Deep learning for generic object detection: A survey. *Int. J. Comput. Vis.*, 128(2):261–318, 2020.

[50] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Proc. Int. Conf. Neural Inf. Process. Syst. (NeurIPS)*, page 91–99, 2015.

[51] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look

once: Unified, real-time object detection. In *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 779–788, 2016.

[52] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. In *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 7263–7271, 2017.

[53] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[54] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single shot multibox detector. In *Proc. Eur. Conf. Comput. Vis. (ECCV)*, pages 21–37, 2016.

[55] Laura Leal-Taixé. Multiple object tracking with context awareness. *arXiv preprint arXiv:1411.7935*, 2014.

[56] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *Proc. IEEE Int. Conf. Image Process. (ICIP)*, pages 3464–3468, 2016.

[57] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *Proc. IEEE Int. Conf. Image Process. (ICIP)*, pages 3645–3649, 2017.

[58] Paul Voigtlaender, Michael Krause, Aljosa Osep, Jonathon Luiten, Berin Balachandar Gnana Sekar, Andreas Geiger, and Bastian Leibe. MOTS: Multi-object tracking and segmentation. In *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 7942–7951, 2019.

[59] Zhongdao Wang, Liang Zheng, Yixuan Liu, Yali Li, and Shengjin Wang. Towards real-time multi-object tracking. In *Proc. Eur. Conf. Comput. Vis. (ECCV)*, pages 107–122, 2020.

[60] Xingyi Zhou, Vladlen Koltun, and Philipp Krähenbühl. Tracking objects as points. In *Proc. Eur. Conf. Comput. Vis. (ECCV)*, pages 474–490, 2020.

[61] Yifu Zhang, Chunyu Wang, Xinggang Wang, Wenjun Zeng, and Wenyu Liu. FairMOT: On the fairness of detection and re-identification in multiple object tracking. *Int. J. Comput. Vis.*, 129:3069–3087, 2021.

[62] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 44(7):3523–3542, 2021.

[63] Shanhe Yi, Zijiang Hao, Qingyang Zhang, Quan Zhang, Weisong Shi, and Qun Li. LAVEA: Latency-aware video analytics on edge computing platform. In *Proc. IEEE/ACM Symp. Edge Compt. (SEC)*, pages 1–13, 2017.

[64] Haoxin Wang, BaekGyu Kim, Jiang Xie, and Zhu Han. LEAF+AIO: Edge-assisted energy-aware object detection for mobile augmented reality. *IEEE Trans. Mobile Comput.*, 22(10):5933–5948, 2022.

[65] Ruoyu Zhang, Yutao Zhou, Fangxin Wang, and Zhi Wang. Maxim: DRL-based cross-camera streaming configuration for real-time video analytics. In *Proc. IEEE Int. Conf. Multimed. Expo (ICME)*, pages 01–06, 2022.

[66] Rui Lu, Chuang Hu, Dan Wang, and Jin Zhang. Gemini: A real-time video

analytics system with dual computing resource control. In *Proc. IEEE/ACM Symp. Edge Compt. (SEC)*, pages 162–174, 2022.

[67] Miao Zhang, Fangxin Wang, and Jiangchuan Liu. CASVA: Configuration-adaptive streaming for live video analytics. In *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, pages 2168–2177, 2022.

[68] Lei Zhang, Yuqing Zhang, Ximing Wu, Fangxin Wang, Laizhong Cui, Zhi Wang, and Jiangchuan Liu. Batch adaptative streaming for video analytics. In *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, pages 2158–2167, 2022.

[69] Peng Yang, Feng Lyu, Wen Wu, Ning Zhang, Li Yu, and Xuemin Sherman Shen. Edge coordinated query configuration for low-latency and accurate video analytics. *IEEE Trans. Ind. Inform.*, 16(7):4855–4864, 2019.

[70] Miaomiao Liu, Xianzhong Ding, and Wan Du. Continuous, real-time object detection on mobile devices without offloading. In *Proc. IEEE Int. Conf. Distrib. Comput. Syst. (ICDCS)*, pages 976–986, 2020.

[71] Xukan Ran, Haolianz Chen, Xiaodan Zhu, Zhenming Liu, and Jiasi Chen. DeepDecision: A mobile deep learning framework for edge video analytics. In *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, pages 1421–1429, 2018.

[72] Xiangyu Li, Yuanchun Li, Yuanzhe Li, Ting Cao, and Yunxin Liu. FlexNN: Efficient and adaptive DNN inference on memory-constrained edge devices. In *Proc. Annu. Int. Conf. Mobile Comput. Netw. (MobiCom)*, pages 709–723, 2024.

[73] Boyuan Feng, Yuke Wang, Gushu Li, Yuan Xie, and Yufei Ding. Palleon:

A runtime system for efficient video processing toward dynamic class skew. In *Proc. USENIX Conf. Annu. Tech. Conf. (USENIX ATC)*, pages 427–441, 2021.

[74] Lin Sun, Weijun Wang, Tingting Yuan, Liang Mi, Haipeng Dai, Yunxin Liu, and Xiaoming Fu. BiSwift: Bandwidth orchestrator for multi-stream video analytics on edge. In *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, pages 1181–1190, 2024.

[75] Sibendu Paul, Kunal Rao, Giuseppe Coviello, Murugan Sankaradas, Oliver Po, Y Charlie Hu, and Srimat Chakradhar. Enhancing video analytics accuracy via real-time automated camera parameter tuning. In *Proc. Conf. Embed. Netw. Sens. Syst. (SenSys)*, pages 291–304, 2022.

[76] Mike Wong, Murali Ramanujam, Guha Balakrishnan, and Ravi Netravali. MadEye: Boosting live video analytics accuracy with adaptive camera configurations. In *Proc. USENIX Symp. Netw. Syst. Des. Implement. (NSDI)*, pages 549–568, 2024.

[77] Manjiri A Namjoshi and Prasad A Kulkarni. Novel online profiling for virtual machines. In *Proc. ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environ.*, pages 133–144, 2010.

[78] Ben Zhang, Xin Jin, Sylvia Ratnasamy, John Wawrzynek, and Edward A Lee. AWStream: Adaptive wide-area streaming analytics. In *Proc. Conf. ACM Special Interest Group Data Comm. (SIGCOMM)*, pages 236–252, 2018.

[79] Ran Xu, Chen-lin Zhang, Pengcheng Wang, Jayoung Lee, Subrata Mitra, Somali Chaterji, Yin Li, and Saurabh Bagchi. ApproxDet: content and contention-aware approximate object detection for mobiles. In *Proc. Conf. Embed. Netw. Sens. Syst. (SenSys)*, pages 449–462, 2020.

[80] Ran Xu, Jayoung Lee, Pengcheng Wang, Saurabh Bagchi, Yin Li, and Somali Chaterji. LiteReconfig: Cost and content aware reconfiguration of video object detection systems for mobile GPUs. In *Proc. Eur. Conf. Comput. Syst. (EuroSys)*, pages 334–351, 2022.

[81] Sheng Zhang, Can Wang, Yibo Jin, Jie Wu, Zhuzhong Qian, Mingjun Xiao, and Sanglu Lu. Adaptive configuration selection and bandwidth allocation for edge-based video analytics. *IEEE/ACM Trans. Netw.*, 30(1):285–298, 2021.

[82] Ning Chen, Siyi Quan, Sheng Zhang, Zhuzhong Qian, Yibo Jin, Jie Wu, Wenzhong Li, and Sanglu Lu. Cuttlefish: Neural configuration adaptation for video analysis in live augmented reality. *IEEE Trans. Parallel Distrib. Syst.*, 32(4):830–841, 2020.

[83] Christopher Canel, Thomas Kim, Giulio Zhou, Conglong Li, Hyeontaek Lim, David G Andersen, Michael Kaminsky, and Subramanya Dulloor. Scaling video analytics on constrained edge nodes. *Proc. Mach. Learn. Syst.*, 1:406–417, 2019.

[84] Junjue Wang, Ziqiang Feng, Zhuo Chen, Shilpa George, Mihir Bala, Padmanabhan Pillai, Shao-Wen Yang, and Mahadev Satyanarayanan. Bandwidth-efficient live video analytics for drones via edge computing. In *Proc. IEEE/ACM Symp. Edge Compt. (SEC)*, pages 159–173, 2018.

[85] Vinod Nigade, Lin Wang, and Henri Bal. Clownfish: Edge and cloud symbiosis for video stream analytics. In *Proc. IEEE/ACM Symp. Edge Compt. (SEC)*, pages 55–69, 2020.

[86] Ting Li, Jiyan Sun, Yinlong Liu, Xu Zhang, Dali Zhu, Zhaorui Guo, and Liru Geng. ESMO: Joint frame scheduling and model caching for edge video analytics. *IEEE Trans. Parallel Distrib. Syst.*, 34(8):2295–2310, 2023.

[87] Jiansheng Dong, Jingling Yuan, Lin Li, Xian Zhong, and Weiru Liu. Optimizing queries over video via lightweight keypoint-based object detection. In *Proc. Int. Conf. Multimedia Retrieval*, pages 548–554, 2020.

[88] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proc. ACM Conf. Embedded Netw. Sensor Syst. (SenSys)*, pages 155–168, 2015.

[89] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. BranchyNet: Fast inference via early exiting from deep neural networks. In *Proc. Int. Conf. Pattern Recognit. (ICPR)*, pages 2464–2469, 2016.

[90] Santosh Kumar Nukavarapu, Mohammed Ayyat, and Tamer Nadeem. iBranchy: An accelerated edge inference platform for IoT devices. In *Proc. IEEE/ACM Symp. Edge Compt. (SEC)*, pages 392–396, 2021.

[91] Lanlan Liu and Jia Deng. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. In *Proc. AAAI Conf. Artif. Intell. (AAAI)*, pages 3675–3682, 2018.

[92] Shiqi Jiang, Zhiqi Lin, Yuanchun Li, Yuanchao Shu, and Yunxin Liu. Flexible high-resolution object detection on edge devices with tunable latency. In *Proc. Annu. Int. Conf. Mobile Comput. Netw. (MobiCom)*, pages 559–572, 2021.

[93] Zheng Yang, Xu Wang, Jiahang Wu, Yi Zhao, Qiang Ma, Xin Miao, Li Zhang, and Zimu Zhou. EdgeDuet: Tiling small object detection for edge assisted autonomous mobile vision. *IEEE/ACM Trans. Netw.*, 31(4):1765–1778, 2022.

[94] Xianwei Lv, Qianqian Wang, Chen Yu, and Hai Jin. A feedback-driven DNN inference acceleration system for edge-assisted video analytics. *IEEE Trans. Comput.*, 72(10):2902–2912, 2023.

[95] Xingwang Wang, Muzi Shen, and Kun Yang. On-edge high-throughput collaborative inference for real-time video analytics. *IEEE Internet Things J.*, 11(20):33097–33109, 2024.

[96] Haosong Peng, Wei Feng, Hao Li, Yufeng Zhan, Qihua Zhou, and Yuanqing Xia. Arena: A patch-of-interest ViT inference acceleration system for edge-assisted video analytics. *arXiv preprint arXiv:2404.09245*, 2024.

[97] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proc. Int. Conf. Mach. Learn. (ICML)*, pages 5156–5165, 2020.

[98] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[99] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. In *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2021.

[100] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. DynamicViT: Efficient vision transformers with dynamic token sparsification. *Adv. Neural Inf. Process. Syst. (NeurIPS)*, 34:13937–13949, 2021.

[101] Yifei Liu, Mathias Gehrig, Nico Messikommer, Marco Cannici, and Davide Scaramuzza. Revisiting token pruning for object detection and instance segmentation. In *Proc. IEEE/CVF Winter Conf. Appl. Comput. Vis. (WACV)*, pages 2658–2668, 2024.

[102] Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your ViT but faster. In *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2023.

[103] Xuwei Xu, Sen Wang, Yudong Chen, Yanping Zheng, Zhewei Wei, and Jiajun Liu. GTP-ViT: efficient vision transformers via graph-based token propagation. In *Proc. IEEE/CVF Winter Conf. Appl. Comput. Vis. (WACV)*, pages 86–95, 2024.

[104] Anton Milan, Laura Leal-Taixé, Ian Reid, Stefan Roth, and Konrad Schindler. MOT16: A benchmark for multi-object tracking. *arXiv preprint arXiv:1603.00831*, 2016.

[105] Tatjana Chavdarova, Pierre Baqué, Stéphane Bouquet, Andrii Maksai, Cijo

Jose, Timur Bagautdinov, Louis Lettry, Pascal Fua, Luc Van Gool, and François Fleuret. WildTrack: A multi-camera hd dataset for dense unscripted pedestrian detection. In *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 5030–5039, 2018.

[106] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics YOLOv8. `https://github.com/ultralytics/ultralytics`, 2024. Accessed: September 9, 2025.

[107] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 248–255, 2009.

[108] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *Proc. Eur. Conf. Comput. Vis. (ECCV)*, pages 740–755, 2014.

[109] Keivan Nalaie, Renjie Xu, and Rong Zheng. DeepScale: Online frame size adaptation for multi-object tracking on smart cameras and edge servers. In *Proc. IEEE/ACM Seventh Int. Conf. Internet-of-Things Design Implement. (IoTDI)*, pages 67–79, 2022.

[110] Renjie Xu, Keivan Nalaie, and Rong Zheng. FastTuner: Fast resolution and model tuning for multi-object tracking in edge video analytics. *IEEE Trans. Mobile Comput.*, 24(6):4747–4761, 2025.

[111] Peigen Ye, Wenfeng Wang, Bing Mi, and Kongyang Chen. EdgeStreaming:

Secure computation intelligence in distributed edge networks for streaming analytics. *ACM Trans. Multimed. Comput. Commun. Appl. (TOMM)*, 2024.

[112] Mingjin Zhang, Jiannong Cao, Yuvraj Sahni, Qianyi Chen, Shan Jiang, and Lei Yang. Blockchain-based collaborative edge intelligence for trustworthy and real-time video surveillance. *IEEE Trans. Ind. Informat.*, 19(2):1623–1633, 2022.

[113] Yuxin Kong, Peng Yang, and Yan Cheng. Adaptive on-device model update for responsive video analytics in adverse environments. *IEEE Trans. Circuits Syst. Video Technol.*, 35(1):857–873, 2025.

[114] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. Deep layer aggregation. In *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 2403–2412, 2018.

[115] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. CenterNet: Keypoint triplets for object detection. In *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, pages 6569–6578, 2019.

[116] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[117] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 4700–4708, 2017.

[118] Keivan Nalaie and Rong Zheng. AttTrack: Online deep attention transfer for multi-object tracking. In *Proc. IEEE/CVF Winter Conf. Appl. Comput. Vis. (WACV)*, pages 1654–1663, 2023.

[119] Neiwen Ling, Kai Wang, Yuze He, Guoliang Xing, and Daqi Xie. RT-mDL: Supporting real-time mixed deep learning tasks on edge platforms. In *Proc. ACM Conf. Embedded Netw. Sensor Syst. (SenSys)*, pages 1–14, 2021.

[120] Fengwei Yu, Wenbo Li, Quanquan Li, Yu Liu, Xiaohua Shi, and Junjie Yan. POI: Multiple object tracking with high performance detection and appearance feature. In *ECCV 2016 Workshops*, pages 36–42, 2016.

[121] Hei Law and Jia Deng. CornerNet: Detecting objects as paired keypoints. In *Proc. Eur. Conf. Comput. Vis. (ECCV)*, pages 734–750, 2018.

[122] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, pages 2980–2988, 2017.

[123] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 7482–7491, 2018.

[124] Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. Real-time video analytics: The killer app for edge computing. *Computer*, 50(10):58–67, 2017.

[125] Tianxiang Tan and Guohong Cao. Deep learning video analytics through edge computing and neural processing units on mobile devices. *IEEE Trans. Mobile Comput.*, 22(3):1433–1448, 2023.

[126] Vinod Nigade, Ramon Winder, Henri Bal, and Lin Wang. Better never than late: Timely edge video analytics over the air. In *Proc. ACM Conf. Embedded Netw. Sensor Syst. (SenSys)*, pages 426–432, 2021.

[127] Biyi Fang, Xiao Zeng, and Mi Zhang. NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proc. Annu. Int. Conf. Mobile Comput. Netw. (MobiCom)*, pages 115–127, 2018.

[128] Royson Lee, Stylianos I. Venieris, Lukasz Dudziak, Sourav Bhattacharya, and Nicholas D. Lane. MobiSR: Efficient on-device super-resolution through heterogeneous mobile processors. In *Proc. Annu. Int. Conf. Mobile Comput. Netw. (MobiCom)*, pages 1–16, 2019.

[129] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Nikolaos Karianakis, Kevin Hsieh, Paramvir Bahl, and Ion Stoica. Ekya: Continuous learning of video analytics models on edge compute servers. In *Proc. USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, pages 119–135, 2022.

[130] Mehrdad Khani, Ganesh Ananthanarayanan, Kevin Hsieh, Junchen Jiang, Ravi Netravali, Yuanchao Shu, Mohammad Alizadeh, and Victor Bahl. RECL: Responsive resource-efficient continuous learning for video analytics. In *Proc. USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, pages 917–932, 2023.

[131] Shreshth Tuli, Giuliano Casale, and Nicholas R Jennings. SplitPlace: AI augmented splitting and placement of large-scale neural networks in mobile edge environments. *IEEE Trans. Mobile Comput.*, 22(9):5539–5554, 2022.

[132] Shusen Yang, Zhanhua Zhang, Cong Zhao, Xin Song, Siyan Guo, and Hailiang Li. CNNPC: End-edge-cloud collaborative CNN inference with joint model partition and compression. *IEEE Trans. Parallel Distrib. Syst.*, 33(12):4039–4056, 2022.

[133] Yubin Duan and Jie Wu. Optimizing job offloading schedule for collaborative DNN inference. *IEEE Trans. Mobile Comput.*, 23(4):3436–3451, 2023.

[134] Xiao Zeng, Biyi Fang, Haichen Shen, and Mi Zhang. Distream: scaling live video analytics with workload-adaptive distributed edge intelligence. In *Proc. ACM Conf. Embedded Netw. Sensor Syst. (SenSys)*, pages 409–421, 2020.

[135] Miao Zhang, Fangxin Wang, Yifei Zhu, Jiangchuan Liu, and Zhi Wang. Towards cloud-edge collaborative online video analytics with fine-grained serverless pipelines. In *Proc. ACM Multimedia Syst. Conf. (MMSys)*, pages 80–93, 2021.

[136] Zhihe Zhao, Kai Wang, Neiwen Ling, and Guoliang Xing. EdgeML: An automl framework for real-time deep learning on the edge. In *Proc. Int. Conf. Internet-of-Things Design Implement. (IoTDI)*, pages 133–144, 2021.

[137] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Comput. Archit. News*, 45(1):615–629, 2017.

[138] Chuang Hu, Wei Bao, Dan Wang, and Fengming Liu. Dynamic adaptive DNN

surgery for inference acceleration on the edge. In *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, pages 1423–1431, 2019.

[139] Jong Hwan Ko, Taesik Na, Mohammad Faisal Amir, and Saibal Mukhopadhyay. Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms. In *Proc. IEEE Int. Conf. Adv. Video Signal Based Surveillance (AVSS)*, pages 1–6, 2018.

[140] Thaha Mohammed, Carlee Joe-Wong, Rohit Babbar, and Mario Di Francesco. Distributed inference acceleration with adaptive DNN partitioning and offloading. In *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, pages 854–863, 2020.

[141] Weiyu Ju, Dong Yuan, Wei Bao, Liming Ge, and Bing Bing Zhou. DeepSave: Saving DNN inference during handovers on the edge. In *Proc. IEEE/ACM Symp. Edge Compt. (SEC)*, pages 166–178, 2019.

[142] Ke-Jou Hsu, Ketan Bhardwaj, and Ada Gavrilovska. Couper: DNN model slicing for visual analytics containers at the edge. In *Proc. IEEE/ACM Symp. Edge Compt. (SEC)*, pages 179–194, 2019.

[143] Mengyuan Chao, Radu Stoleru, Liuyi Jin, Shuochao Yao, Maxwell Maurice, and Roger Blalock. AMVP: Adaptive CNN-based multitask video processing on mobile stream processing platforms. In *Proc. IEEE/ACM Symp. Edge Compt. (SEC)*, pages 96–109, 2020.

[144] Jian He, Chenxi Yang, Zhaoyuan He, Ghufran Baig, and Lili Qiu. Scheduling DNNs on edge servers. *arXiv preprint arXiv:2304.09961*, 2023.

[145] Kichang Yang, Juheon Yi, Kyungjin Lee, and Youngki Lee. FlexPatch: Fast and accurate object detection for on-device high-resolution live video analytics. In *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, pages 1898–1907, 2022.

[146] Tianxiang Tan and Guohong Cao. Deep learning on mobile devices through neural processing units and edge computing. In *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, pages 1209–1218, 2022.

[147] Ayan Chakrabarti, Roch Guérin, Chenyang Lu, and Jiangnan Liu. Real-time edge classification: Optimal offloading under token bucket constraints. In *Proc. IEEE/ACM Symp. Edge Compt. (SEC)*, pages 41–54, 2021.

[148] Kaifei Chen, Tong Li, Hyung-Sin Kim, David E Culler, and Randy H Katz. Marvel: Enabling mobile augmented reality with low energy and low latency. In *Proc. ACM Conf. Embedded Netw. Sensor Syst. (SenSys)*, pages 292–304, 2018.

[149] Luyang Liu and Marco Gruteser. EdgeSharing: Edge assisted real-time localization and object sharing in urban streets. In *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, pages 1–10, 2021.

[150] Ganesh Ananthanarayanan, Yuanchao Shu, Landon Cox, and Victor Bahl. Project Rocket platform—designed for easy, customizable live video analytics—is open source. `https://www.microsoft.com/en-us/research/publication/project-rocket-platform-designed-/for-easy-customizable-live-video-analytics-is-open-source/`, 2022. Accessed: September 9, 2025.

[151] Anurag Ghosh, Srinivasan Iyengar, Stephen Lee, Anuj Rathore, and Venkat N Padmanabhan. Streaming video analytics on the edge with asynchronous cloud support. *arXiv preprint arXiv:2210.01402*, 2022.

[152] Keivan Nalaie and Rong Zheng. MVSparse: Distributed cooperative multi-camera multi-target tracking on the edge. In *Proc. IEEE Int. Conf. Adv. Video Signal Based Surveillance (AVSS)*, pages 1–7, 2024.

[153] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Forward-backward error: Automatic detection of tracking failures. In *Proc. Int. Conf. Pattern Recognit. (ICPR)*, pages 2756–2759, 2010.

[154] Wei Liu, Shengcai Liao, Weiqiang Ren, Weidong Hu, and Yinan Yu. High-level semantic feature detection: A new perspective for pedestrian detection. In *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 5187–5196, 2019.

[155] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proc. IEEE/CVF Conf. Comput. Vision and Pattern Recognit. (CVPR)*, pages 4510–4520, 2018.

[156] Irtiza Hasan, Shengcai Liao, Jinpeng Li, Saad Ullah Akram, and Ling Shao. Generalizable pedestrian detection: The elephant in the room. In *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 11328–11337, June 2021.

[157] Sucheng Ren, Fangyun Wei, Zheng Zhang, and Han Hu. TinyMIM: An empirical study of distilling mim pre-trained models. In *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 3687–3697, 2023.

[158] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open MMLab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019.

[159] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[160] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[161] Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, et al. A survey on vision transformer. *IEEE Trans. Pattern Anal. Mach. Intell.*, 45(1):87–110, 2022.

[162] Linyi Jiang, Silvery D Fu, Yifei Zhu, and Bo Li. Janus: Collaborative vision transformer under dynamic network environment. In *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, pages 1–10, 2025.

[163] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross

Girshick. Masked autoencoders are scalable vision learners. In *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 16000–16009, 2022.

[164] Byungseok Roh, JaeWoong Shin, Wuhyun Shin, and Saehoon Kim. Sparse DETR: Efficient end-to-end object detection with learnable sparsity. In *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2022.

[165] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.