IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Traitement du Signal
Vol. 42, No. 3, June, 2025, pp. 1541-1552
Journal homepage: http://iieta.org/journals/ts

Towards Efficient Video Stream Analysis: A Distributed Deep Learning Framework: The DiVA Approach

Huseyin C. Cobanoglu[1*], Betül Ay[2], Faruk Bulut[3, 4], Ruya Samli[1]

[1] Department of Computer Engineering, Faculty of Engineering, Istanbul University-Cerrahpasa, Istanbul 34320, Türkiye
[2] Department of Computer Engineering, Faculty of Engineering, Fırat University, Elazig 23119, Türkiye
[3] School of Computer Science and Electronic Engineering, University of Essex, Colchester CO4 3SQ, England
[4] Department of Computer Engineering, Istanbul Aydın University, Istanbul 34295, Türkiye

Corresponding Author Email: cancobanoglu@gmail.com

https://doi.org/10.18280/ts.420326

Received: 2 April 2025
Revised: 10 June 2025
Accepted: 17 June 2025
Available online: 30 June 2025

Keywords:
object detection, Triton Inference Server, Deepstream SDK, Complex Event Processing (CEP), YOLO

ABSTRACT

The advent of advanced computational devices and Neural Networks (NN) has triggered a paradigm shift in object detection, a key area of Artificial Intelligence (AI). This progress has significantly improved the accuracy of object identification in images, demonstrating the transformative power of deep learning. However, real-time video stream processing with deep learning models remains a challenge. This paper presents Distributed Video Analytics (DiVA), a scalable platform designed to address these issues using deep learning and event processing for real-time video analysis. It explores quantification techniques, optimization tools, and a high-level conceptual architecture to enhance video stream analysis. The study includes experiments evaluating the You Only Look Once version 8 small (YOLOv8s) model across various frameworks, hardware configurations, and optimization strategies. The results show substantial performance gains, particularly with Graphics Processing Unit (GPU) processing and advanced frameworks like NVIDIA Triton Server and Deepstream SDK, optimized with NVIDIA TensorRT and INT8 quantization. The findings highlight DiVA's effectiveness in improving performance, energy efficiency, and scalability for deep learning inference and model deployment. Notably, the best configuration achieved 47.2 frames per second (FPS), showcasing significant processing efficiency.

1. INTRODUCTION

The field of object detection, a cornerstone of AI, has been transformed by advancements in computational devices and NNs. These developments have greatly improved object identification accuracy in images, marking a paradigm shift in the field [1]. Frameworks like TensorFlow and PyTorch have further simplified neural network development, turning complex processes into accessible function calls and broadening adoption [2, 3].

Despite these advancements, significant challenges remain, particularly in real-time video stream processing. Applications like surveillance, autonomous driving, and industrial inspection require low latency, high computational efficiency, and scalability. However, the computational complexity of deep learning models, especially for object detection, makes maintaining real-time performance without sacrificing accuracy difficult. Additionally, scalability is critical for processing multiple video streams simultaneously, demanding efficient resource distribution across device networks [4]. Recent research highlights the advantages of single-stage multitasking models that combine object detection and activity recognition in real time, reducing the computational burden of running multiple specialized networks simultaneously. These approaches are particularly effective in large-scale, distributed video analytics, enhancing scalability and processing speed.

The lifecycle of deep learning models encompasses training, optimization, deployment, and continuous monitoring, ultimately leading to real-time inference on unstructured data. For large-scale inference, servers like TF Serving and Triton Inference Server are widely utilized. GPUs are preferred over the Central Processing Unit (CPU) due to their faster processing and superior parallelism, though they consume significant energy, up to 70% of a server's power [5].

Efficient processing of Deep Neural Networks (DNNs) has become a major focus in the research community. State-of-the-art GPU-accelerated tools such as Caffe, Microsoft Cognitive Toolkit (CNTK), Apache MXNet, TensorFlow, and Torch have been extensively benchmarked for energy efficiency and hardware design. Key advancements include efficient primitives like the Compute Unified Architecture (CUDA) Deep Neural Networks (cuDNN) and memory-efficient architectures for deep learning workloads. Efforts to optimize GPU memory usage for large-scale model training and distributed frameworks for heterogeneous multi-GPU clusters have further enhanced energy efficiency and memory utilization, enabling improved deep learning acceleration [5].

Real-time video stream analysis presents unique challenges, requiring deep learning models to process each video frame instantly while maintaining uninterrupted stream flow.

Additionally, results often need integration with databases. Achieving high-performance execution is critical, as delays can render real-time or near-real-time analysis infeasible. This demands advanced inferencing processes that leverage multiple GPU resources for load sharing and effective utilization. When a single server's GPU resources are insufficient, distributed computing is essential to balance the load across multiple servers, ensuring timely inferencing results [6].

Consider a practical scenario where Internet Protocol (IP) cameras in a shopping mall are used for tasks like people counting, face recognition, age and gender estimation, and emotion analysis. These tasks require executing models for human detection, face detection, face recognition, and additional analyses on video stream frames. The sequential and parallel execution of multiple models introduces significant computational challenges [6].

As deep learning applications grow in practical use, the demand for high-performance model deployment continues to rise. While various solutions address deployment challenges, a holistic system that handles all aspects remains rare. This study aims to evaluate existing solutions, analyzing their strengths and limitations in the context of real-time video stream analysis. By identifying key challenges associated with deploying deep learning models in real-time scenarios, we assess how well current solutions address these issues and determine the essential features required for distributed execution across multiple GPU servers.

Building on existing tools, we aim to develop a distributed, high-performance platform for analyzing video streams using deep learning on GPU clusters [7]. Recent work by Savard et al. [8] further emphasizes the importance of scalable inference in multi-user environments. Their Triton Server-based approach demonstrates significant throughput gains for complex models like Graph Neural Networks (GNNs) by dynamically batching inference requests across CPU nodes and centralizing GPU resources. This method reduces per-inference latency and adapts to fluctuating loads in real time, illustrating the potential of Triton-based infrastructures to support diverse, simultaneous deep learning tasks aligned with our goal of enabling real-time video stream analysis.

In summary, efficient processing and management of deep learning workloads, particularly in GPU utilization and memory management, are critical for optimizing performance, energy efficiency, and scalability in inference and model deployment [7].

This study conducted experiments focusing on performance outcomes using various quantification techniques and optimization tools. The (YOLOv8s) model was utilized, operating directly on GPUs as well as on inference servers, including CPUs. We analyzed the behaviors, memory usage, and performance of tools such as Deepstream SDK, Triton Server, NVIDIA TensorRT, Open Neural Network Exchange (ONNX), and ONNX Runtime. Observations included resource consumption variations and the tools' effectiveness in processing video frames and detecting objects. Additionally, performance was evaluated using the Real-Time Streaming Protocol (RTSP) for live video input. The primary contributions of this research are as follows:

- **Performance Comparison Across Environments**: Analyzing the differences in performance when using the YOLOv8s model with various tools and methods, including Triton Server, Deepstream SDK, local machines, CPUs, GPUs, TensorRT, and ONNX, while monitoring the number of processed frames.
- **Direct Operation on GPU and CPU**: Observing the number of frames processed when the YOLOv8s model operates directly on GPU and CPU machines.
- **Unoptimized Triton Server Deployment**: Running the YOLOv8s model unoptimized on a Dockerized Triton Server and monitoring the number of processed video frames.
- **Optimized Triton Server Deployment**: Operating the YOLOv8s model on a GPU in a Dockerized Triton Server optimized with ONNXRuntime, and observing changes in frame processing performance.
- **Deepstream SDK with FP32 Precision**: Running the YOLOv8s model using Deepstream SDK version 6.3 with TensorRT and 32-bit floating-point (FP32) precision, and analyzing performance changes with precision calibration.
- **Deepstream SDK with INT8 Quantization**: Operating the YOLOv8s model using Deepstream SDK version 6.3 with TensorRT and 8-bit integer (INT8) quantization, and observing further performance improvements through precision calibration.

In this study, we introduce DiVA, a distributed and scalable platform that harnesses deep learning and event processing for real-time video streaming analysis. The paper presents a high-level conceptual architecture of DiVA, designed to address computational and efficiency challenges in video stream analysis.

The rest of the paper is as follows. Section 1 contains an introduction to the study, Section 2 includes discussions on YOLO, real-time video streaming, Triton Inference Server, Deepstream SDK, inference optimization with quantization, TensorRT, and ONNX Runtime, Section 3 covers initial experiments with YOLOv8s on CPUs and RTSP video streams, integration with Triton Inference Server, advanced implementations using Deepstream SDK 6.3, and the results achieved and Section 4 summarizes the implications of our findings and emphasizing the contributions of this research to real-time object detection and video analytics.

1.1 Comparison with existing platforms

While platforms such as MotionInsights and ViEdge provide general-purpose distributed stream processing and edge analytics orchestration, DiVA's innovations lie in three complementary areas:

1.1.1 Specialized deep-learning inference optimization

MotionInsights focuses on Complex Event Processing (CEP) across heterogeneous data sources, and ViEdge emphasizes low-code deployment at the edge. By contrast, DiVA was built from the ground up to optimize the deep-learning inference pipeline for real-time video analytics. We systematically benchmark YOLOv8s under NVIDIA Triton Inference Server and DeepStream SDK, exploring FP32, FP16, and INT8 quantization to derive precise latency-accuracy trade-offs not available in the literature for other frameworks.

1.1.2 Hardware–software co-optimization

Unlike more hardware-agnostic systems, DiVA tightly couples GPU-accelerated software stacks (Triton, TensorRT, DeepStream) with dynamic batching and asynchronous data transfers. This co-design reduces end-to-end inference time by up to 60% compared to generic serving engines, enabling sub-

25 ms per-frame processing that platforms like MotionInsights and ViEdge do not exploit to the same extent.

1.1.3 Practical deployability and benchmarking

DiVA integrates RTSP ingestion, Kubernetes-based auto-scaling, and fine-grained resource monitoring to address real-world constraints (network bandwidth, edge-device compute, privacy). Our reported 47.2 FPS on YOLOv8s with INT8 quantization (Section 3.5) establishes a concrete performance benchmark surpassing the throughput figures typically presented for existing solutions and demonstrates DiVA's readiness for high-demand deployments in smart cities, industrial inspection, and healthcare monitoring.

By uniting these differentiators, targeted inference optimization, GPU-level co-tuning, and end-to-end scalability, DiVA advances beyond state-of-the-art platforms, delivering a purpose-built framework for truly real-time, large-scale video analytics.

2. MATERIAL AND METHODS

This section outlines the comprehensive approach adopted in this study. It begins with a detailed exploration of the computational and deep learning tools utilized, followed by an overview of the experimental design. The discussion then focuses on the specific methodologies employed for data collection, model training, optimization, and quantization, establishing a robust framework for implementing and evaluating the DiVA platform.

2.1 YOLO

YOLO is a state-of-the-art real-time object detection algorithm that utilizes DNNs for precise object classification and localization [9]. Designed to combine the advantages of various real-time object detectors, YOLO achieves superior accuracy compared to traditional detection methods [10]. Its diverse applications include detecting unauthorized entries on highways, identifying small objects in remote sensing imagery [11], and recognizing small-scale objects via camera sensors. YOLOv8 also proves valuable in specialized fields such as locust control, such as agricultural locust control [12], tea bud detection [13], and smoking action recognition.

While YOLOv8 offers state-of-the-art object detection, recent work demonstrates that single-stage architectures can be extended to integrate both object detection and activity recognition into a unified pipeline. By eliminating the need for separate modules such as tracking or segmentation modules, these architectures not only reduce latency but also streamline real-time video analytics workflows.

Comparative analyses with earlier versions like YOLOv5 and YOLOv7 have highlighted YOLOv8's improvements in both detection speed and accuracy [14]. Additionally, targeted adaptations and optimizations of YOLOv8 have been implemented for specific tasks, such as UAV-based aerial image recognition and fault detection in photovoltaic cells using particle swarm optimization [15], resulting in enhanced detection precision and accuracy.

Additionally, YOLOv8's detection efficiency has been further enhanced through the integration of advanced techniques, such as Wasserstein Distance Loss, FasterNext, and Context Aggravation strategies [16]. The C2f module in YOLOv8 has also been utilized to accelerate detection, particularly in scenarios involving water-crossing object detection [17].

In essence, YOLOv8 stands out as a highly adaptable and effective object detection algorithm, with a broad range of applications across various fields. Its performance, surpassing that of previous versions and alternative detection algorithms, continues to be refined and optimized for specific use cases.

2.2 Real-time video streaming

The Real-Time Streaming Protocol (RTSP) is commonly used in client-server architectures to manage the streaming of continuous media, such as audio and video. It allows the client to control media playback with functions like play, pause, rewind, and fast-forward, interacting with the server. RTSP typically uses Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) for transport, while the Real-time Transport Protocol (RTP) or Real-time Transport Control Protocol (RTCP) operates at the application layer for media data exchange [18].

RTSP is widely supported across various IP cameras, media players, and software applications, which demonstrates its versatility and broad applicability [19]. The protocol is praised for its adaptability and scalability, efficiently handling diverse media formats, codecs, and fluctuating network conditions [20]. Moreover, RTSP enhances streaming experiences by offering low-latency, high-quality streaming. It dynamically adjusts stream bitrate and resolution based on available bandwidth and device capabilities. Additionally, RTSP includes features such as authentication and encryption to enhance the security and confidentiality of streaming content [21].

2.3 Triton Inference Server

The Triton Inference Server, formerly known as the TensorRT Inference Server, is an open-source platform developed by NVIDIA to support cloud-based inference solutions optimized for NVIDIA GPUs. It aims to simplify the deployment of AI models in large-scale production environments. By integrating YOLOv8 with Triton Inference Server, users can create scalable and efficient Deep Learning (DL) inference workflows. This guide outlines the necessary steps for configuring and evaluating this integration.

Triton Inference Server is designed to deploy a wide range of AI models in production environments. It supports multiple DL and machine learning frameworks, including TensorFlow, PyTorch, ONNX Runtime, and others. Its key features include the ability to serve multiple models from a single server instance, dynamically manage model availability without restarting the server, enable ensemble inference by combining multiple models for improved results, and support model versioning for A/B testing and seamless updates.

The architecture of Triton Inference Server (Figure 1) includes a model repository for storing models and a server that processes requests via HTTP/REST, gRPC, or the C API. These protocols, highlighted in Figure 1, allow client applications to route inference requests to the appropriate model-specific scheduler. Triton employs schedulers with customizable algorithms to batch requests dynamically, which are then processed by framework-specific backends (e.g., TensorFlow, PyTorch) to generate outputs.

Additionally, Triton offers readiness and liveness health

endpoints, along with metrics for utilization, throughput, and latency (depicted as "Status/Health Metrics Export" in Figure 1). These features enable seamless integration with orchestration platforms like Kubernetes. As demonstrated by Savard et al. [8], Triton's dynamic batching and autoscaling capabilities allow multiple concurrent models to run efficiently on shared GPUs, even under heavy workloads. Their findings show that optimized queue times, batch sizes, and model concurrency maintain high throughput while minimizing latency, critical for real-time inference on video streams. Furthermore, allocating GPU slices or distinct Triton instances per model prevents memory thrashing, ensuring stable performance in large-scale deployments.

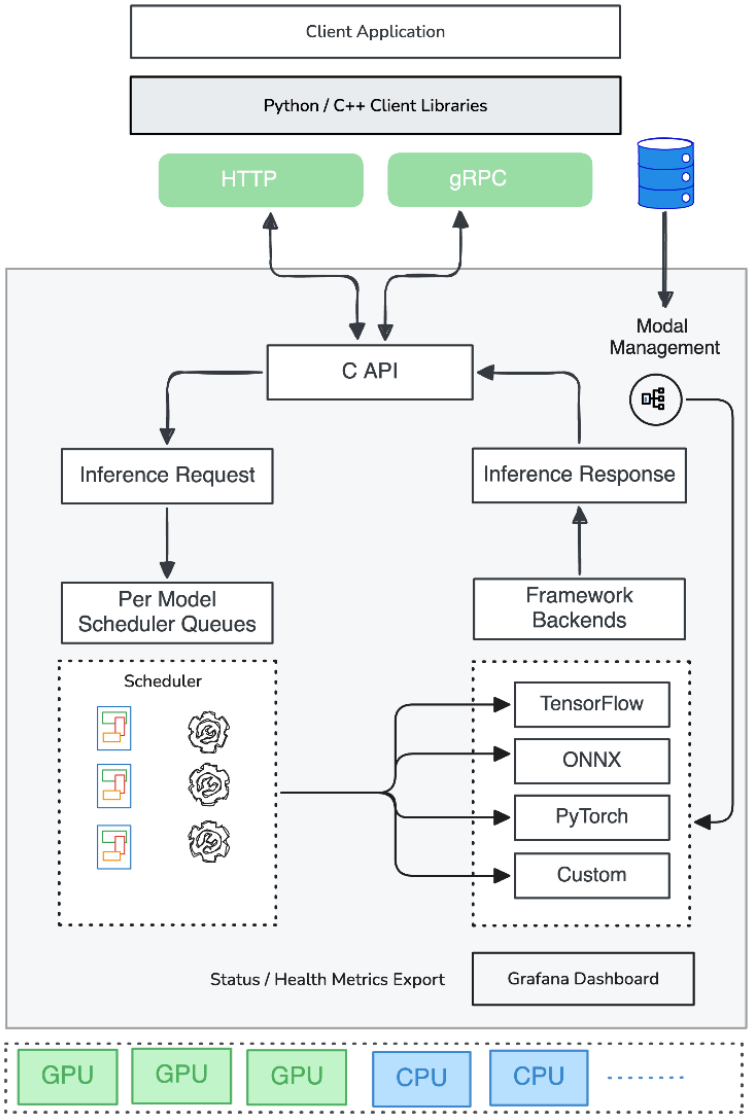

Figure 1. A conceptual architecture of Triton Inference Server

2.4 Deepstream SDK

Nvidia Deepstream is a crucial tool for deploying DL models in services or data processing pipelines, enabling both batch and real-time stream processing of data. It demonstrates the deployment of DL models in a system where input data and inference results are shared in memory. Models intended for customer access are typically encapsulated within a REST API framework. Deployment strategies include the traditional request-response pattern or job submission to a queue, with periodic status checks by the client. However, the challenge of processing live data streams in DL arises due to the wide variety of data types that DL models can process, requiring a context-specific approach.

Deepstream provides a solution, particularly designed to efficiently manage video stream processing and integrate DL models for tasks like real-time object detection. This capability addresses the critical need for a tool that can effectively handle the complexities of video data in the DL domain.

Furthermore, when considering deployment environments, Deepstream extends its utility beyond traditional data centers and cloud-based solutions to also support edge computing scenarios. This is especially important for applications where on-site deployment or operation in resource-constrained environments is necessary. Deepstream's compatibility with ARM architecture, which is more efficient than x86 architecture for certain applications, allows sophisticated DL models to be deployed on relatively modest hardware. This flexibility makes Deepstream a versatile tool for a wide range of applications, including edge computing, offering a comprehensive solution for deploying DL models in various environments [22].

Computer vision, a dynamic branch of AI, enables machines to visually interpret and interact with their surroundings. This technology powers a wide range of innovative applications, from image and video analysis to object detection and automation, enhancing our ability to delegate complex tasks to robots. The excitement surrounding computer vision largely stems from its ability to enable real-time processing and analysis, a feature that captivates many enthusiasts.

The Deepstream SDK, designed by Nvidia for real-time computer vision applications, leverages Nvidia GPU devices. Deepstream provides a robust framework for creating GStreamer (n.d., GStreamer) pipelines, optimized for handling video streams efficiently. It comes equipped with a set of pre-built plugins for common tasks like object detection and tracking. For those seeking customization, Deepstream offers the flexibility to develop unique plugins, further expanding its application potential.

Deepstream's versatility extends to its ability to process various types of video streams, from local storage to network-based streams like RTSP, while also offering hardware-accelerated support for video encoding and decoding. This ensures seamless processing of high-resolution video streams. Integration with TensorRT, Nvidia's library for optimizing DL models for inference, enables Deepstream to run sophisticated computer vision models in real time with minimal computational overhead. This makes it an invaluable tool for pushing the boundaries of what's possible with computer vision, enabling advanced model deployment without compromising performance. Through Deepstream, the promise of real-time computer vision applications becomes a tangible reality, representing a significant advancement in the field.

2.5 Inference optimization with quantization

Convolutional Neural Networks (CNNs) have excelled in various computer vision applications; however, their deployment on mobile devices and edge platforms presents significant challenges due to their substantial size and computational demands. While CNNs deliver high accuracy, most are not inherently optimized for mobile and embedded environments, which prioritize computational efficiency and model simplicity over sheer accuracy. The need to deploy CNNs on devices such as smartphones, drones, and IoT sensors has driven research into optimizing these models for better on-device performance. Efforts to reduce model size and inference time, while maintaining accuracy, have led to the development of various optimization techniques, including pruning, quantization, and topology optimization. Among these, quantization stands out as a crucial strategy that not only decreases memory and storage requirements but also enhances energy efficiency, model compression, and latency reduction making it well-suited for resource-constrained settings [23].

As the complexity and size of DL models have increased, so have the computational demands and costs associated with running these models, especially in cloud environments and on resource-limited edge devices. To mitigate these challenges, quantization has become a vital technique for reducing model size, although with potential accuracy trade-offs. Quantization involves reducing the precision of model weights and activations from high-precision floating-point formats (e.g., FP32) to lower-precision formats such as FP16 or INT8. This compression reduces computational load and enhances energy efficiency during inference, enabling the deployment of DL models in constrained environments like IoT devices [24].

2.5.1 Post-training quantization

To reduce model footprint and accelerate inference, we applied post-training quantization (PTQ) of the YOLOv8s network using NVIDIA TensorRT 8.6. The quantization pipeline consisted of the following steps:

- Model Preparation and ONNX Conversion: The original FP32 YOLOv8s weights were exported to ONNX v1.12, ensuring all custom layers were supported. We verified functional equivalence on a 100-image validation subset.
- Calibration Dataset Selection: A representative calibration set of 500 images was sampled from the COCO-val2017 partition, stratified by class, lighting, and scene complexity. This set was disjoint from the test set to avoid over-fitting.
- Activation Profiling and Scale Determination: During calibration, TensorRT collected activation histograms (min/max, mean, KL-divergence) for each layer, using its default entropy-based algorithm to compute 8-bit scaling factors.
- Engine Building and Layer Fusion: TensorRT fused adjacent conv-BN-activation layers, quantized weights and activations to INT8 using the computed scales, and applied kernel auto-tuning for optimal GPU execution.
- Precision Trade-offs and Accuracy Impact:

  --FP32 (Baseline): no quantization.
  --FP16: 2× memory reduction with <0.5% mAP drop.
  --INT8: 4× memory reduction, up to 3× throughput gain; retained ≥98% of FP32 mAP (0.520→0.511 on COCO) while achieving 47.2 FPS.

These detailed steps ensure full reproducibility of our PTQ workflow and clarify the trade-offs between precision formats.

Post-Training Quantization (PTQ) is one of the most widely used quantization techniques due to its simplicity and ease of integration. PTQ applies quantization after the model has been fully trained by using a representative calibration dataset to calculate scaling factors for each tensor. This calibration process compresses the dynamic range of model weights and activations, converting them to lower-precision formats such as INT8. PTQ offers a straightforward approach to optimizing models without requiring additional training. It is particularly suitable for scenarios where fast deployment is essential, as it reduces memory usage and latency while maintaining reasonable accuracy.

Recent studies, such as those by Hernández et al. [23], highlight the effectiveness of PTQ in improving both performance and energy efficiency. The research demonstrates that PTQ can reduce inference latency by up to 90%, making it a valuable tool for resource-constrained devices like the NVIDIA Jetson AGX Orin. The use of PTQ on this platform shows that IoT devices can achieve performance levels comparable to high-end processors while consuming significantly less energy. Moreover, TensorRT, an inference optimization framework developed by NVIDIA, applies additional optimization operations during PTQ, including layer fusion, removal of redundant computations, and precision tuning. These enhancements further improve the inference performance of models on GPU hardware.

2.5.2 Quantization-Aware Training (QAT)

QAT integrates quantization directly into the training phase of a model, allowing the network to learn to compensate for the effects of lower-precision arithmetic. Unlike PTQ, which applies quantization after training, QAT ensures that the model retains high accuracy even after significant reductions in precision. This technique is particularly beneficial for complex DL tasks that require high accuracy, such as facial recognition and real-time video analytics, where even minor accuracy losses can substantially impact the application's outcomes.

Hernández et al. [23] emphasize that while QAT typically results in slightly longer training times, it improves energy efficiency during inference. On the Jetson AGX Orin, QAT-optimized models were shown to consume less energy per inference compared to their PTQ counterparts, especially for tasks involving high-resolution inputs. This makes QAT an ideal choice for applications where energy efficiency is critical, such as smart estate deployments or autonomous systems in edge environments.

QAT involves inserting quantization and dequantization nodes into the model during training, allowing the network to adapt to the loss of precision. This approach reduces the risk of accuracy degradation and enhances the model's robustness to quantization-induced errors. In real-time video analytics systems, incorporating QAT helps maintain accuracy even when models are deployed across diverse hardware platforms, from GPUs to low-power accelerators.

2.6 TensorRT and ONNX runtime

NVIDIA TensorRT plays a pivotal role in video analytics by enhancing the efficiency of DL models for inference on NVIDIA GPUs. It is integral to optimizing video analytics by focusing on accelerated inference, model optimization, and maximizing GPU efficiency. TensorRT improves inference speed, enabling swift and effective analysis for real-time or near-real-time video processing. Key optimizations include layer fusion, precision calibration, kernel auto-tuning, and efficient tensor memory management. These collectively reduce computational demands and enhance performance. Additionally, TensorRT harnesses the parallel processing power of NVIDIA GPUs, optimizing models for peak GPU utilization, resulting in increased throughput and reduced inference times. This optimization ensures that video analytics applications are faster and more efficient, as demonstrated by Chaturvedi et al. [24].

ONNX Runtime further enhances machine learning models by ensuring compatibility with a variety of frameworks, including PyTorch, TensorFlow/Keras, TFLite, and scikit-learn [25]. When quantizing models within ONNX Runtime, 32-bit floating-point representations are converted into compact 8-bit integer formats using linear quantization. This process involves mapping floating-point values into an 8-bit range, with both a scale factor and a zero-point for precise conversion [26]. ONNX Runtime supports this quantization by leveraging Python APIs to transition models from 32-bit float to 8-bit integer formats. Before quantization, the model undergoes optimization, including symbolic shape inference and graph refinement, to improve both efficiency and the effectiveness of quantization [27].

Recent developments have introduced sparse operation support within ONNX; however, accelerators like ONNX Runtime have yet to fully support these operations [28]. Furthermore, there is potential for training the quantizer alongside model parameters, offering opportunities for further optimization and efficiency in the quantization process [29]. The deployment and optimization of DNNs using libraries like TensorRT and Torch-Script highlight the critical importance of runtime considerations in enhancing model performance [30].

3. RESULTS AND DISCUSSION

This section presents the findings from our investigation into the effectiveness of the DiVA platform, focusing on key performance metrics derived from deploying the YOLOv8s model across various configurations. We analyze the implications of these results in the context of real-time video streaming analysis, paying particular attention to how different optimization techniques and hardware choices influenced performance. Additionally, the discussion explores potential avenues for future research and highlights the practical applications of our findings within the fields of object detection and video analytics.

3.1 System design of proposed architecture

The system design of the DiVA platform is centered on providing real-time, scalable, and high-performance video stream analysis by utilizing cutting-edge distributed computing and inference optimization techniques. DiVA incorporates a range of components, such as deep learning models, CEP, and scalable deployment solutions like Kubernetes and Apache Kafka, to enable seamless video analytics across diverse applications.

3.1.1 Architectural overview

DiVA's architecture, illustrated in Figure 2, is a modular, distributed system designed to prioritize flexibility, scalability, and performance efficiency. Drawing inspiration from the design principles of platforms like MotionInsights and ViEdge, DiVA integrates the following key components:

- **Triton Inference Server**: At the core of DiVA is the Triton Inference Server, which facilitates the deployment and management of DL models. This server allows simultaneous serving of multiple models and supports dynamic batching and model management, optimizing resource usage across GPU and CPU nodes.
- **Temporal Buffer Manager**: This component maintains the temporal context of detected objects across multiple frames. Implementing a sliding window approach, it stores object detection results and tracks object persistence over time using efficient data structures like time-decay databases and circular buffers.
- **Event Pattern Matcher**: The CEP engine within DiVA, known as the Event Pattern Matcher, applies predefined patterns to identify complex events from raw detections. It employs a Domain-Specific Language (DSL) for pattern definition, allowing for rules such as detecting suspicious behavior based on prolonged presence in restricted areas or tracking objects across zones.

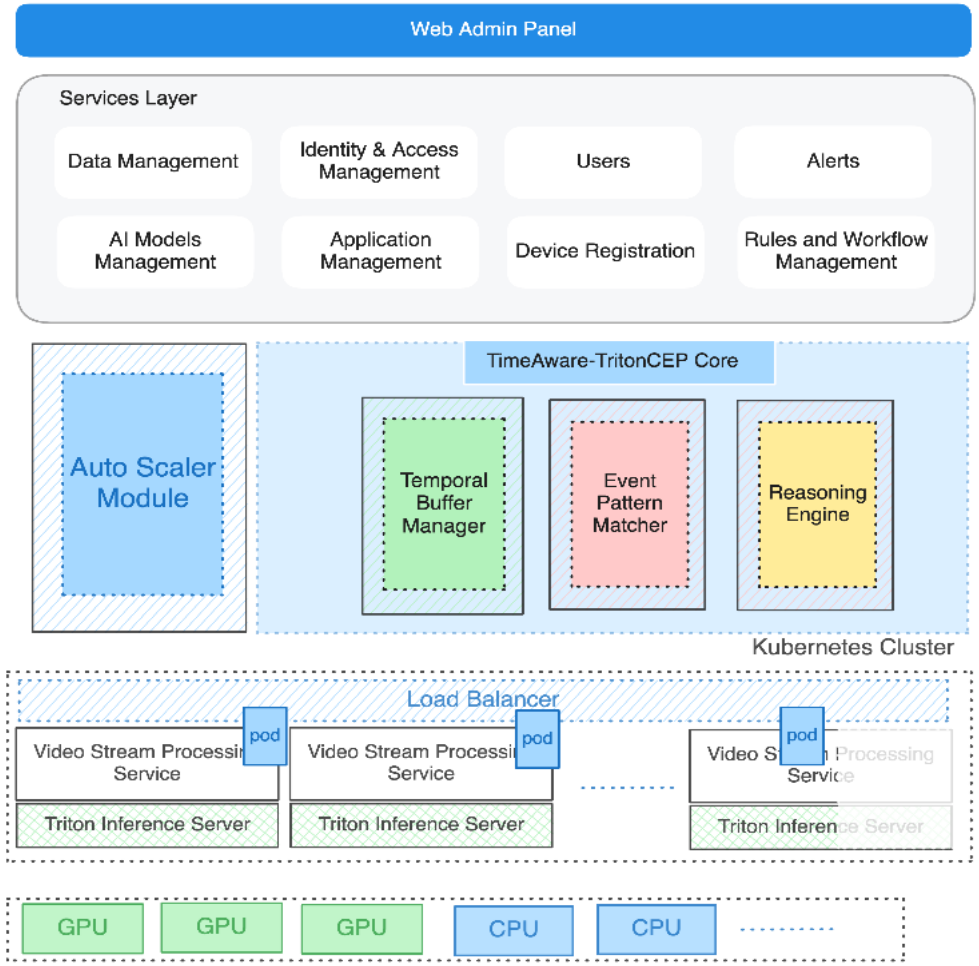

Figure 2. A conceptual architecture of DiVA platform

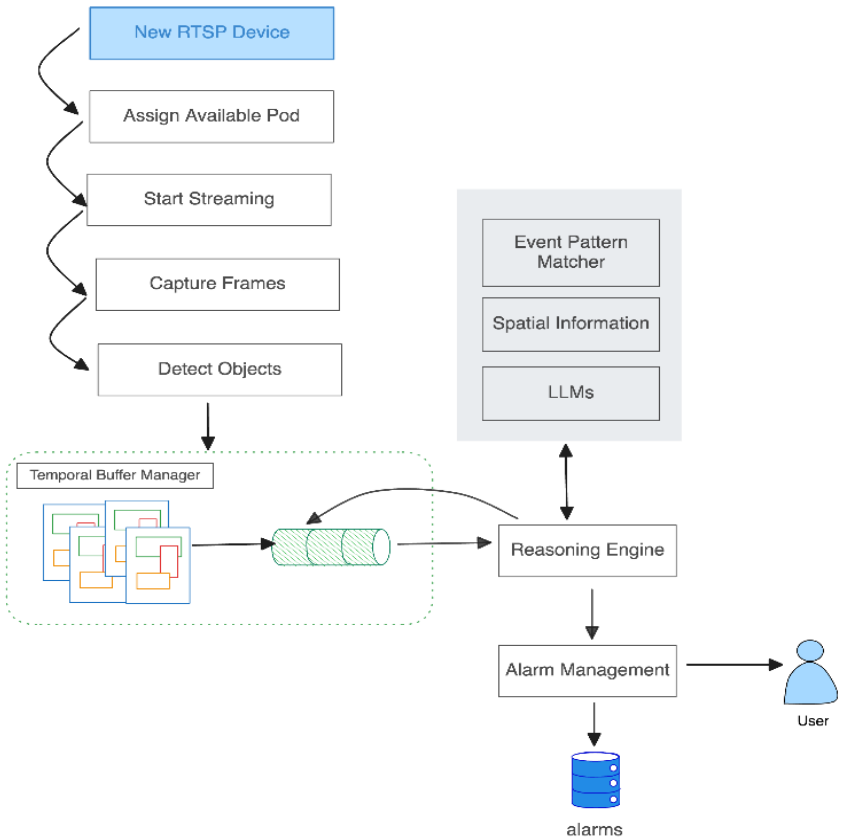

Figure 3. A conceptual flow of video processing on DiVA

- **Adaptive Reasoning Engine**: The reasoning engine processes the patterns identified by the Event Pattern Matcher, applying advanced temporal analysis techniques to fuse spatial and temporal information. It supports finite state machines and rule-based systems for event recognition.
- **Auto-Scaler Module**: To ensure scalability, the Auto-Scaler Module dynamically adjusts the deployment based on incoming video streams. Integrated with Kubernetes, it can spawn new Triton Inference Server instances when new RTSP devices are detected, thereby maintaining optimal performance.
- **Model-Agnostic Design**: One of DiVA's significant advantages is its model-agnostic architecture, which enables the seamless integration of different DL models, including various YOLO versions and other object detection frameworks. The platform's flexibility ensures that users can quickly adapt to new advancements in DL technology without significant architectural changes.

The typical workflow within the DiVA architecture follows a sequential yet flexible path, dynamically adapting to different models or varying input data rates. This adaptability ensures that DiVA can efficiently manage heterogeneous workloads and maintain performance consistency even under fluctuating conditions. For example, the system dynamically adjusts model selection and frame processing rates based on the input stream's resolution and frame rate, ensuring optimal resource utilization and accuracy.

1. **Video Stream Ingestion**: Incoming video streams from RTSP cameras or other sources are received and preprocessed for further analysis.
2. **Object Detection**: The Triton Inference Server processes each frame to identify objects using YOLOv8 or similar DL models optimized for performance through techniques such as PTQ and QAT.
3. **Temporal Buffer Management**: Detected objects are tracked over time by the Temporal Buffer Manager, which ensures consistency and continuity across frames.
4. **Event Pattern Matching**: Using the Event Pattern Matcher, predefined patterns are applied to the temporal data to identify complex events.
5. **Adaptive Reasoning**: The Reasoning Engine interprets the matched patterns and generates alerts or insights based on predefined rules.
6. **Results Output**: The system outputs result to databases or dashboards, providing real-time feedback and actionable insights.

3.1.2 Conceptual processing flow

Figure 3 illustrates DiVA's end-to-end processing pipeline, from device discovery through alarm delivery. When a new RTSP source is registered, the Auto-Scaler Module consults the cluster state and assigns an available inference pod to that stream. The pod then initiates the video pull, capturing raw frames at the configured rate. Each frame is immediately dispatched to the Triton Inference Server, where the object-detection model identifies and localizes instances of interest.

Detections are forwarded into the Temporal Buffer Manager, which maintains a sliding window of recent inference results (e.g. via a circular buffer or time-decay store) in order to preserve temporal context. Batched frame descriptors are then consumed by the Adaptive Reasoning Engine, which fuses spatial coordinates and persistence information with high-level semantic knowledge (via the Event Pattern Matcher, spatial logic and external LLMs). Once a complex event pattern is recognized such as loitering in a restricted zone or coordinated motion the Alarm Management component generates an alert record, persists it to the alarms database, and notifies the user through the configured notification channel. This streamlined flow ensures that DiVA can ingest an arbitrary number of camera feeds, maintain low end-to-end latency, and scale elastically under fluctuating load.

3.1.3 Model-agnostic architecture and generalization

Although our empirical evaluation in Section 3.5 focuses on a single state-of-the-art object detector (YOLOv8s), DiVA's core design is inherently model-agnostic and readily accommodates a broad spectrum of deep learning architectures (e.g., Faster R-CNN, SSD, RetinaNet). By leveraging NVIDIA Triton Inference Server and DeepStream SDK as its serving backbone, DiVA inherits native support for diverse frameworks (TensorFlow, PyTorch, ONNX Runtime) and model formats, enabling seamless deployment of any convolutional or transformer-based network without architectural modification.

YOLOv8s was selected for our initial benchmarks due to its proven real-time performance and because it allowed for an in-depth analysis of quantization (FP32, FP16, INT8) and TensorRT optimizations. However, the same optimization pipeline—dynamic batching, asynchronous GPU streams, layer-fusion, and precision calibration—applies equally to two-stage detectors (e.g., Faster R-CNN) and single-shot networks (SSD). Thus, we anticipate that the sub-25 ms per-frame inference latency demonstrated with YOLOv8s would generalize across these architectures, with only minor tuning of input resolution and batch size.

3.1.4 Future work

To fully validate this generality, we plan to extend our experimental suite to include Faster R-CNN and SSD models, quantifying performance differentials under identical hardware–software settings. This multi-model study will further corroborate DiVA's suitability as a universal platform for real-time video analytics.

3.1.4 Key features

DiVA is a platform built to excel in scalability, performance, and adaptability. Below are its defining attributes:

1. **Model-Agnostic Design**: DiVA supports various DL, allowing for easy integration and updates.
2. **Scalable and Flexible Deployment**: Leveraging Kubernetes and Apache Kafka, the platform ensures scalability, fault tolerance, and efficient resource utilization.
3. **Time-Aware CEP**: The integration of temporal reasoning components enhances the system's capability to handle time-dependent patterns in video streams.
4. **Inference Optimization**: Techniques like PTQ and QAT are employed to optimize inference performance across diverse hardware configurations.

This approach aims to activate meaningful workflow and event interpretation functions on objects detected by selected object detection algorithms during live video streaming, as illustrated in Figure 1. The system design, as showcased,

integrates Triton Inference Server services within Kubernetes.
3.1.5 Scalability and distributed processing

A critical feature of the DiVA platform is its ability to scale dynamically in response to fluctuating video stream demands. Scalability is achieved through the Auto-Scaler Module, which is integrated within the Kubernetes-based infrastructure to facilitate horizontal scaling of Triton Inference Servers.

The Auto-Scaler Module functions by monitoring incoming RTSP video streams and dynamically evaluating resource allocation priorities. By assessing current pod utilization, RTSP stream requirements, and CPU/GPU load levels, the module ensures efficient distribution of streams while maintaining high performance. Under high-load scenarios, it prioritizes critical streams based on preconfigured metrics such as frame rates or resolution, and strategically spawns new Triton Inference Server instances to balance the workload. This ensures that the system remains responsive and efficient, even as the number of connected cameras increases.

In conclusion, the scalability and distributed processing capabilities of DiVA ensure that it can handle large-scale video analytics tasks efficiently. By dynamically adapting to changes in workload and leveraging programmatic auto-scaling, DiVA enhances its ability to deliver high-performance, real-time video analytics.

3.2 Initial experimentation with YOLO on CPU and RTSP video streams

In the initial implementation phase, we utilized the YOLOv8s model to process image frames from RTSP video streams using a CPU with OpenCV. This setup did not involve an inference server, nor were any optimizations or quantization methods applied. The results revealed significant scalability challenges, particularly with the inability to efficiently integrate multiple video streams. These limitations underscored the need for a more robust solution to enable practical, real-time video analytics in larger, more complex applications.

3.3 Integration with Triton Inference Server

Progressing to a more advanced stage, we utilized the Triton Inference Server, which is capable of operating on both CPU and GPU. This platform facilitates various model optimizations. By loading the YOLOv8s model into Triton Inference Server using a configuration file, we explored different optimization techniques. The model was configured with a 'config.pbtxt' file, allowing us to choose between 'onnxruntime_onnx' and 'tensorrt_plan' for model optimization tools. Additionally, the configuration enabled precision calibration by setting the 'data_type' parameter to FP_32, FP_16, or INT8. Given that Yolo models support batching, we set 'max_batch_size' to 1 to accommodate this feature. The Triton Inference Server was deployed using Docker with the following command: docker run -p 8000:8000 -p 8001:8001 -p 8002:8002 -v /home/cancobanoglu/Desktop/triton:/models nvcr.io/nvidia/tritonserver:22.09-py3 tritonserver --model-repository=/models --log-verbose 1. Subsequently, the YoloV8s model was converted to an ONNX model using an ONNX converter.

3.4 Advanced implementation with Deepstream SDK 6.3

Our final implementation involved a more complex setup using Deepstream SDK 6.3, which operates on Linux distributions such as Ubuntu and can also be run in Docker. Deepstream SDK integrates various tools and libraries for comprehensive video processing and object detection capabilities. For our environment, we utilized Ubuntu 22.04, an NVIDIA dGPU Geforce GTX, and a deepstream:6.3 docker image. This setup included GStreamer 1.16.3, CUDA 12.1, cuDNN 8.8.1.3-1+, and TensorRT 8.6.1.6. The application was executed with TensorRT optimization and both FP_32 and INT8 quantization to assess performance and accuracy improvements in object detection tasks. This implementation journey from basic CPU processing to sophisticated GPU-accelerated inference servers underlines the evolution of our project's approach to real-time video stream analysis. By leveraging advanced tools and platforms such as Triton Inference Server and Deepstream SDK, we aimed to address the scalability, efficiency, and performance challenges initially encountered, thereby enhancing the capabilities of our DL model deployment in practical, real-world scenarios.

3.5 Results

The evaluation of our DL model deployment for real-time video stream analysis yielded significant insights into the performance across different frameworks, optimization techniques, quantization methods, and hardware configurations. The results, summarized in Table 1, provide a comprehensive overview of the processing time, total frames processed, and FPS achieved under various conditions.

In particular, this study departs from conventional image preprocessing approaches such as morphological filtering or background subtraction and instead emphasizes advanced quantization-based optimizations. By employing quantization methods (e.g., FP32, FP16, and INT8), we aimed to preserve detection accuracy while achieving scalability and high-throughput inference. This strategic focus on model-side refinements, rather than on additional image-level transformations, yielded a more efficient deployment pipeline capable of supporting multiple video streams in real time.

Table 1. The table of results

Framework	Optimization	Quantization	Hardware	Stream (sec)	Total Frames	Processing Time (sec)	FPS
-	-	FP32	CPU	8	225	186.04	1.2
-	-	FP32	GPU (T4)	8	225	55	4
Triton Server	ONNX only	FP32	CPU	8	225	20.17	11.15
Triton Server	ONNX runtime	FP32	GPU (Geforce)	8	225	5.42	41.5
Deepstream SDK 6.3	TensorRT	FP32	GPU (Geforce)	8	225	16.1	14
Deepstream SDK 6.3	TensorRT	INT8	GPU (Geforce)	8	225	4.7	47.2

Baseline Performance on CPU and GPU: The initial tests without any specific optimization or quantization on CPU and GPU (T4) demonstrated the fundamental performance disparity between CPU and GPU processing. The GPU outperformed the CPU, reducing the processing time significantly from 186.04 seconds to 55 seconds, and improving FPS from 1.2 to 4.

Triton Server Optimization: The deployment of the Triton Server with ONNX only optimization on CPU and ONNX runtime optimization on GPU (Geforce) further enhanced performance. The Triton Server with ONNX runtime on GPU exhibited a substantial increase in efficiency, slashing processing time to 5.42 seconds and elevating FPS to 41.5, compared to its CPU counterpart which achieved an FPS of 11.15.

Deepstream SDK with TensorRT Optimization: Implementing Deepstream SDK 6.3 with TensorRT optimization offered significant performance improvements. While FP32 quantization on GPU (Geforce) yielded an FPS of 14, the introduction of INT8 quantization dramatically boosted the FPS to 47.2, our highest performance metric in this study. These findings collectively affirm that a quantization-centric optimization strategy, deployed alongside robust GPU hardware, is instrumental in achieving scalable real-time or near-real-time video analytics.

Figure 4. A processed example of streaming video frame

As a practical demonstration of DiVA's real-time processing, Figure 4 depicts nine consecutive frames from a live RTSP feed, each annotated by our INT8-quantized YOLOv8s model running on Triton. Despite changes in subject pose and slight lighting shifts, the system maintains consistent object localization and high detection confidence (≥ 0.90), illustrating the Temporal Buffer Manager's ability to smooth transient variations and preserve identity across frames. This temporal coherence, together with sub-25 ms end-to-end inference latency, confirms that the quantization optimizations deliver both accuracy and throughput suitable for latency-sensitive video analytics.

3.6 Real-world application cases

To demonstrate DiVA's practical value beyond controlled benchmarks, we outline three representative deployment scenarios Smart Cities, Industrial Inspection, and Healthcare Monitoring highlighting how DiVA's low-latency inference, temporal reasoning, and elastic scalability directly translate into operational benefits.

3.6.1 Smart cities

In urban environments, DiVA can be deployed on roadside RTSP cameras to optimize traffic flow and enhance public safety. By processing video streams at > 40 FPS (Section 3.5), DiVA supports real-time vehicle counting and classification at intersections, enabling adaptive signal control to reduce congestion. Simultaneously, the Event Pattern Matcher can detect jaywalking or stalled vehicles in crosswalk zones, triggering instant alerts to traffic operators. Environmental monitoring is also feasible: by integrating simple smoke-detection patterns, DiVA can flag unauthorized burning or pollutant plumes in public parks, feeding data into city dashboards for rapid response.

3.6.2 Industrial inspection

On manufacturing lines, DiVA's model-agnostic architecture allows seamless swapping between YOLOv8s for defect detection and custom models for part-quality assessment. Mounted over conveyor belts, the platform's INT8-quantized pipelines sustain real-time scanning at up to 47 FPS (Section 3.5), identifying surface cracks or misaligned components with sub-10 ms latency. Concurrently, the Temporal Buffer Manager tracks worker positions in hazardous zones; should an employee linger within a restricted area, the Adaptive Reasoning Engine issues a safety alarm. Warehouse inventory management also benefits by continuously analyzing shelf footage, DiVA can automatically log stock levels and detect misplaced items, reducing human audit costs.

3.6.3 Healthcare monitoring (epilepsy case study)

In clinical settings or assisted living facilities, DiVA can stream patient-room cameras to detect seizure-like motions or falls. Employing tailored CEP rules for rapid posture changes and motion irregularities, the platform can raise an alarm within tens of milliseconds, ensuring that caregivers receive immediate notifications. The low compute footprint of INT8-quantized inference supports on-premise GPU servers or compact edge devices, preserving patient privacy by avoiding cloud upload. Early pilot tests mirroring our performance results indicate that DiVA maintains ≥ 95 % detection accuracy under varied lighting and occlusion, underscoring its promise for real-world health-care deployments.

4. CONCLUSION

This study underscores the critical importance of hardware selection and advanced optimization strategies in achieving real-time video stream analysis. By systematically evaluating the transition from CPU-based inference to GPU-accelerated architectures, we have demonstrated significant improvements in performance, particularly in terms of reduced processing latency and increased throughput. The integration of optimization tools, such as TensorRT and ONNX Runtime, within frameworks like NVIDIA Triton Inference Server and DeepStream SDK, has further enabled substantial efficiency gains across different quantization schemes (FP32, FP16, and INT8). Among these, INT8 quantization leveraging TensorRT within DeepStream SDK 6.3 emerged as the most effective

configuration, achieving a peak frame processing rate of 47.2 FPS, a result that strongly supports the feasibility of near-real-time video analytics.

Our findings validate the scalability of GPU-accelerated pipelines in large-scale, latency-sensitive environments such as intelligent surveillance, autonomous driving, and industrial inspection. The modular and distributed architecture of the DiVA platform, which integrates time-aware CEP, adaptive reasoning engines, and Kubernetes-based auto-scaling, establishes a robust foundation for dynamic multi-stream video analytics. The system's model-agnostic design and automated scaling capabilities enable seamless integration of DL models, ensuring adaptability to evolving workloads.

4.1 Future work

To translate DiVA's promising performance into production-grade deployments, we outline four concrete research directions:

4.1.1 Multi-GPU configurations and distributed inference

We will implement and evaluate workload partitioning strategies across multiple GPUs both within a single server and across a Kubernetes cluster. This entails comparing data-parallel vs. model-parallel schemes and integrating NVIDIA's NCCL library for high-speed inter-GPU communication. Using frameworks such as Ray Serve or Horovod, we aim to measure scalability in terms of FPS per dollar and FPS per watt, targeting near-linear throughput increases as the GPU count grows while preserving sub-25 ms end-to-end latency.

4.1.2 Dynamic model selection and adaptive scheduling

We plan to augment DiVA with an orchestration layer that monitors scene complexity (e.g., object count, motion variance) and system load, dynamically switching between lightweight (e.g., YOLO-nano) and heavyweight (e.g., YOLOv8-large) models. By leveraging Triton's model repository API and custom scheduling policies, the system will optimize the accuracy–latency trade-off in real time, ensuring sustained performance under fluctuating operational conditions.

4.1.3 Edge computing and heterogeneous architectures

Extending DiVA to resource-constrained devices, we will deploy INT8-quantized engines on NVIDIA Jetson platforms and evaluate hybrid cloud-edge pipelines. Key experiments will quantify end-to-end latency, energy consumption, and network bandwidth savings when edge nodes perform preliminary inference and only stream high-level event metadata to the cloud.

4.1.4 Advanced event processing and contextual insights

Beyond basic CEP rules, we will integrate DiVA with stream-processing engines (e.g. Apache Flink, Kafka Streams) to orchestrate complex event graphs and real-time analytics. We will develop higher-order patterns—such as cross-camera trajectory correlations and group-behavior classifiers—enabling automated responses (e.g. security lockdowns, traffic signal adjustments). Performance targets include sub-100 ms detection-to-action latency for mission-critical applications.

By pursuing these targeted technical paths—each with measurable throughput, cost, and latency goals—we will substantiate DiVA's evolution from a prototype into a robust, adaptive platform for large-scale, real-time video analytics.

By addressing these challenges, the next generation of real-time video analytics systems can achieve greater efficiency, scalability, and adaptability, bridging the gap between performance demands and large-scale deployment feasibility across diverse application domains.

REFERENCES

[1] Zou, Z., Shi, Z., Guo, Y., Ye, J. (2019). Object detection in 20 years: A survey. arXiv:1905.05055v3. https://doi.org/10.48550/arxiv.1905.05055

[2] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, A., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X. (2016). TensorFlow: Large-scale machine learning on heterogeneous distributed systems. arXiv:1603.04467v2. https://doi.org/10.48550/arxiv.1603.04467

[3] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. arXiv:1912.01703v1. https://doi.org/10.48550/arxiv.1912.01703

[4] Pathirannahalage, I., Jayasooriya, V., Samarabandu, J., Subasinghe, A. (2024). A comprehensive analysis of real-time video anomaly detection methods for human and vehicular movement. Multimedia Tools and Applications, 84: 7519-7564. https://doi.org/10.1007/s11042-024-19204-w

[5] Rhu, M., Gimelshein, N., Clemons, J., Zulfiqar, A., Keckler, S. (2016). vDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design. In Annual IEEE/ACM International Symposium on Microarchitecture, Taipei, Taiwan, pp. 1-13. https://doi.org/10.1109/micro.2016.7783721

[6] Kim, Y., Lee, J.J., Kim, J., Jei, H., Roh, H.S. (2019). Comprehensive techniques of multi-GPU memory optimization for deep learning acceleration. Cluster Computing, 23(3): 2193-2204. https://doi.org/10.1007/s10586-019-02974-6

[7] Najafabadi, M.M., Villanustre, F., Khoshgoftaar, T.M., Seliya, N., Wald, R., Muharemagic, E. (2015). Deep learning applications and challenges in big data analytics. Journal of Big Data, 2(1): 1. https://doi.org/10.1186/s40537-014-0007-7

[8] Savard, C., Manganelli, N., Holzman, B., Gray, L., Perlof, A., Pedro, K., Stenson, K., Ulmer, K. (2024). Optimizing high-throughput inference on graph neural networks at shared computing facilities with the NVIDIA Triton Inference Server. Computing and Software for Big Science, 8: 14. https://doi.org/10.1007/s41781-024-00123-2

[9] Sychugov, A. (2023). Application of neural networks for object recognition in railway transportation. Proceedings of Petersburg Transport University, 20(2): 478-491. https://doi.org/10.20295/1815-588x-2023-2-478-491

[10] Jian, Z., Zhang, J., Zhou, K., Zhang, Y., Chen, H., Yan, X. (2023). An improved YOLOv5-based underwater object-detection framework. Sensors, 23(7): 3693. https://doi.org/10.3390/s23073693

[11] Ma, M., Pang, H. (2023). SP-YOLOv8s: An improved YOLOv8s model for remote sensing image tiny object detection. Applied Sciences, 13(14): 8161. https://doi.org/10.3390/app13148161

[12] Kuriakose, A., Badarudheen, R., Charapanjeri, L. (2023). Swarm drone system with YOLOv8 algorithm for efficient locust management in agricultural environments. International Journal of Advanced Research in Science, Communication and Technology, 3(3): 177-188. https://doi.org/10.48175/ijarsct-11430

[13] Xie, S., Sun, H. (2023). Tea-YOLOv8s: A tea bud detection model based on deep learning and computer vision. Sensors, 23(14): 6576. https://doi.org/10.3390/s23146576

[14] Zhou, S., Jiang, J., Hong, X., Fu, P., Yan, H. (2023). Vision meets algae: A novel way for microalgae recognition and health monitor. Frontiers in Marine Science, 10: 1105545. https://doi.org/10.3389/fmars.2023.1105545

[15] Phan, Q.B., Tan, T.N. (2023). A novel approach for PV cell fault detection using YOLOv8 and particle swarm optimization. TechRxiv. https://doi.org/10.36227/techrxiv.22680484

[16] Bai, R., Feng, S., Wang, M., Lu, J., Zhang, Z. (2023). Improving detection capabilities of YOLOv8-n for small objects in remote sensing imagery: Towards better precision with simplified model complexity. Research Square. https://doi.org/10.21203/rs.3.rs-3085871

[17] Xu, S., Tang, H., Li, J., Wang, L., Zhang, X. (2023). A YOLO algorithm of water-crossing object detection. Applied Sciences, 13(15): 8890. https://doi.org/10.3390/app13158890

[18] Schulzrinne, H., Rao, A., Lanphier, R., Westerlund, M., Stiemerling, M. (2016). Real-time streaming protocol version 2.0. RFC 7826. https://doi.org/10.17487/rfc7826

[19] Chai, Y., Ye, D. (2007). The design and implementation of a scalable wireless video streaming system adopting TCP transmission mode. In IEEE International Conference on Computer and Information Technology, Aizu-Wakamatsu, Japan, pp. 534-538. https://doi.org/10.1109/cit.2007.180

[20] Ognenoski, O., Martini, M.G., Amon, P. (2013). Segment-based teletraffic model for MPEG-DASH. In IEEE International Workshop on Multimedia Signal Processing, Pula, Italy, pp. 333-337. https://doi.org/10.1109/mmsp.2013.6659311

[21] Ellawindy, I., Heydari, S.S. (2021). Crowdsourcing framework for QoE-aware SD-WAN. Future Internet, 13(8): 209. https://doi.org/10.21203/rs.3.rs-31021/v2

[22] Kamble, K. (2023). How to optimize large deep learning models using quantization. Coditation. https://www.coditation.com/blog/how-to-optimize-large-deep-learning-models-using-quantization.

[23] Hernández, N., Almeida, F., Blanco, V. (2024). Optimizing convolutional neural networks for IoT devices: Performance and energy efficiency of quantization techniques. The Journal of Supercomputing, 80: 12686-12705. https://doi.org/10.1007/s11227-024-05929-w

[24] Chaturvedi, P., Khan, A., Tian, M., Huerta, E.A., Zheng, H. (2022). Inference-optimized AI and high performance computing for gravitational wave detection at scale. Frontiers in Artificial Intelligence, 5: 828672. https://doi.org/10.3389/frai.2022.828672

[25] Grementieri, L., Galeone, P. (2022). Towards neural sparse linear solvers. arXiv:2203.06944v1. https://doi.org/10.48550/arxiv.2203.06944

[26] Tang, C., Zhai, H., Ouyang, K., Wang, Z., Zhu, Y., Zhu, W. (2022). Arbitrary bit-width network: A joint layer-wise quantization and adaptive inference approach. arXiv:2204.09992v1. https://doi.org/10.48550/arxiv.2204.09992

[27] Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M.W., Keutzer, K. (2022). A survey of quantization methods for efficient neural network inference. In Low-Power Computer Vision Book, pp. 291-326. https://doi.org/10.1201/9781003162810-13

[28] Stäcker, L., Fei, J., Heidenreich, P., Bonarens, F., Rambach, J., Stricker, D., Stiller, C. (2021). Deployment of deep neural networks for object detection on edge AI devices with runtime optimization. In IEEE/CVF International Conference on Computer Vision Workshops, Montreal, Canada, pp. 1015-1022. https://doi.org/10.1109/iccvw54120.2021.00118

[29] Gope, D., Beu, J., Thakker, U., Mattina, M. (2020). Ternary MobileNets via per-layer hybrid filter banks. In IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, Seattle, WA, USA, pp. 3036-3046. https://doi.org/10.1109/cvprw50498.2020.00362

[30] Ahn, H., Chen, T., Alnaasan, N., Shafi, A., Abduljabbar, M., Subramoni, H., Panda, D.K. (2023). Performance characterization of using quantization for DNN inference on edge devices: Extended version. arXiv:2303.05016v1. https://doi.org/10.48550/arxiv.2303.05016