Distributed edge cloud architecture for executing AI based applications

Youssouph FAYE

August 13, 2025

Université Savoie Mont Blanc

The harmony of the world is made manifest in Form and Number, and the heart and soul and all the poetry of Natural Philosophy are embodied in the concept of mathematical beauty.

– D'Arcy Wentworth Thompson

Preface

I am of the opinion that every LaTeX geek, at least once during his life, feels the need to create his or her own class: this is what happened to me and here is the result, which, however, should be seen as a work still in progress. Actually, this class is not completely original, but it is a blend of all the best ideas that I have found in a number of guides, tutorials, blogs and tex.stackexchange.com posts. In particular, the main ideas come from two sources:

- Ken Arroyo Ohori's Doctoral Thesis, which served, with the author's permission, as a backbone for the implementation of this class;
- The Tufte-Latex Class, which was a model for the style.

The first chapter of this book is introductory and covers the most essential features of the class. Next, there is a bunch of chapters devoted to all the commands and environments that you may use in writing a book; in particular, it will be explained how to add notes, figures and tables, and references. The second part deals with the page layout and design, as well as additional features like coloured boxes and theorem environments.

I started writing this class as an experiment, and as such it should be regarded. Since it has always been intended for my personal use, it may not be perfect but I find it quite satisfactory for the use I want to make of it. I share this work in the hope that someone might find here the inspiration for writing his or her own class.

Federico Marotta

Contents

List of Figures

List of Tables

Introduction 1

1.1 Motivation

1.1.1 The Evolution of Video Analytics and the Need for Edge Computing

The proliferation of video streams has revolutionized various fields, from navigation and security to control systems, by providing an indispensable source of information. Whether it's monitoring traffic for efficient navigation, enhancing security through surveillance, or controlling industrial processes, video streams offer real-time insights that drive decision-making and automation. However, the exponential growth in data volume has made cloud-based processing increasingly impractical. The limitations of bandwidth and latency, coupled with the dependence on centralized resources, create significant bottlenecks. These issues make it difficult to process video data in real-time, leading to delays and inefficiencies.

This has driven a shift towards edge computing, which brings processing closer to the data source. By moving computation to the edge, we can reduce latency, conserve bandwidth, and improve the overall responsiveness of video analytics systems. Edge computing offers numerous benefits, but it also presents unique challenges, particularly in terms of resource constraints. Even with powerful GPUs, resources are not unlimited, compelling us to rethink traditional deployment strategies.

1.1.2 Optimizing Resource Utilization in Edge Computing

One effective approach to optimize resource utilization in edge computing is to decouple the processing pipeline into multiple functions. In live video analytics, for example, each function can process data and send results to the next stage in the pipeline. This modular approach allows for more flexible and efficient use of edge resources. However, existing solutions often deploy the entire pipeline on a single device or split it across resources without adequately considering the interference that co-deployed models can have on each other's performance. This interference can lead to reduced efficiency and increased latency.

To address this, we need to recognize and measure the interference between co-deployed models before deploying them on a target resource. By doing so, we can strategically deploy functions across available resources, maximizing the efficiency of edge devices. This requires a good understanding of how the GPU execution model works during inference. Such an understanding hinges on grasping the internal mechanics of GPU architecture. Designed to manage intensive workloads, GPUs rely on a highly parallel structure made up of multiple streaming multiprocessors, each with its own set of processing cores, registers, and shared memory.

This configuration enables GPUs to excel at the training and inference tasks central to machine learning and deep neural networks (DNNs).

Frameworks like Compute Unified Device Architecture (CUDA) and Open Computing Language (OpenCL) have enhanced GPU utility by offering granular control over kernel execution, memory access, and thread parallelism. CUDA, in particular, has become the standard for programming NVIDIA GPUs, enabling researchers and engineers to harness low-level optimization capabilities and achieve breakthrough AI performance. However, understanding how kernels are scheduled and executed in the GPU during inference is crucial. Each model will eventually execute multiple kernels, and each kernel will require a certain amount of register allocation, shared memory access, and warps to execute efficiently. By optimizing these aspects, we can maximize the performance of AI workloads and ensure efficient processing of video analytics.

1.1.3 Addressing Fluctuating Workloads and Mobile Cameras

Beyond deployment complexities, the performance of video analytics systems is often constrained by fluctuating workloads from camera sources. These temporal imbalances in data generation can cause certain nodes to be overloaded while leaving others underutilized, potentially leading to data gaps and diminished analytical precision. To mitigate this, architects have explored distribution strategies, either by offloading excess traffic vertically between edge devices and centralized cloud resources, or by dispersing workloads horizontally across edge clusters. However, such strategies implicitly rely on the predictability and stability of video input, typically generated from fixed cameras with consistent monitoring patterns.

This assumption begins to break down with the emergence of mobile cameras. Unlike their fixed counterparts, mobile cameras offer advantages such as increased coverage, dynamic perspectives, and enhanced situational awareness. Yet, they also introduce distinct challenges—most notably, the rapid and unpredictable variation in scene content. This volatility disrupts traditional workload planning and demands adaptive processing strategies that can respond in real-time to fluctuating video streams and computational needs.

This thesis aims to introduce and design an innovative architecture tailored to the aforementioned challenges associated with deploying video analytics applications in resource-constrained environments. The proposed framework will establish efficient methods for distributing and managing processing pipelines to adapt to fluctuating workloads. By optimizing resource efficiency and streamlining task distribution, we will be able to improve the overall performance and reliability of video analytics systems in varied and constantly evolving scenarios.

1.2 Contributions of this Thesis

This thesis makes several key contributions to the field of edge computing and video analytics:

Self-Balancing Architecture for Live Video Analytics: VideoJam implements a distributed load balancing system by deploying load balancers co-located with each task in the analytics pipeline. Each load balancer continuously monitors the incoming flow of frames or objects and periodically shares this information with neighboring load balancers. Based on the collected data, each load balancer independently decides how much traffic to process locally and whether to offload some of its workload to less-burdened neighbors using a lightweight machine learning model that predicts the incoming workload for each processing component and its neighbors in the near future. Additionally, the load balancers employ a congestion prevention signaling system to correct any prediction errors. VideoJam operates autonomously, adapting dynamically to changes such as new camera arrivals or departures without requiring system reboots, and balances incoming traffic accordingly. Our approach uniquely combines horizontal distribution and per-function type load balancing, driven by short-term forecasts of incoming loads.

Efficient Model Cohabitation in Edge Computing Model Serving: an orchestration architecture designed to maximize system performance in scenarios where model colocation is necessary, particularly in resource-constrained edge environments. The key innovation of Roomie is its kernel-aware interference profiling, which captures the sequential nature of GPU kernel execution patterns when multiple models share hardware resources. By analyzing how specific kernel sequences from different models interact, Roomie constructs accurate interference profiles that predict performance degradation under various colocation scenarios. This detailed approach allows Roomie to make optimal placement decisions, determining which models can efficiently coexist on the same hardware and which combinations should be avoided to ensure performance guarantees.

Building upon these contributions, the next chapter (Chapter 2) will explore the foundational background and relevant literature that inform our research. This contextual groundwork is essential for grasping the significance of our innovations and the methodological choices made throughout the thesis, thereby enabling a holistic understanding of our strategy for optimizing edge computing and video analytics.

Related work 2

2.1 Edge Computing and Video Analytics

Edge computing represents a transformative shift in distributed computing, wherein computational tasks and data storage are relocated closer to the data source. This paradigm addresses several limitations inherent in centralized cloud architectures, notably by reducing latency, minimizing bandwidth consumption, enhancing data security, and enabling real-time responsiveness. As edge devices—such as sensors, cameras, and IoT nodes—typically operate under constrained computational and energy resources, a variety of architectural models have emerged. These range from hybrid edge-cloud systems, which balance local and remote processing, to fully autonomous edge analytics platforms tailored for specific domains like industrial IoT.

The integration of Graphics Processing Units (GPUs) in edge computing has brought significant advantages. GPUs are designed to handle parallel processing, making them ideal for compute-intensive tasks such as machine learning, computer vision, and data analytics. The use of GPUs in edge computing provides several benefits, including improved performance, increased efficiency, and enhanced scalability. These advantages have paved the way for GPUs to become a cornerstone in enabling advanced edge computing capabilities. By harnessing their parallel processing power, edge systems now support a wide array of real-time applications ranging from autonomous vehicles and smart surveillance to immersive AR/VR experiences and responsive healthcare diagnostics. These applications benefit from edge computing's ability to process data in real-time, reducing latency and improving decision-making. For instance, in IoT, edge computing enables real-time monitoring and control of industrial equipment, improving efficiency and reducing downtime.

Among these applications, video analytics stands out as a particularly demanding and impactful use case [**<empty citation>**]. In contexts such as smart cities, autonomous driving, and public safety, edge-based video analytics enables timely interpretation of visual data. A typical video analytics pipeline comprises several stages: data ingestion, preprocessing, feature extraction, model inference, and post-processing. Each stage imposes distinct computational and memory requirements, which are often challenging to meet within the limited capabilities of edge devices.

These constraints underscore the need for innovative solutions. Edge devices may lack sufficient processing power, storage capacity, or communication bandwidth to support complex video analytics workloads. Consequently, researchers have explored a range of strategies to optimize resource utilization while preserving real-time performance. These include lightweight model architectures, distributed processing frameworks, and adaptive scheduling algorithms that dynamically allocate tasks across heterogeneous edge nodes.

To address these issues, researchers have proposed a spectrum of solutions aimed at optimizing performance while preserving the benefits of edge locality. Techniques such as model compression and pruning reduce the computational footprint of deep learning algorithms, enabling their deployment on resource-constrained devices. Distributed processing frameworks allow workloads to be partitioned and executed collaboratively across multiple edge nodes, thereby enhancing scalability and fault tolerance. Additionally, adaptive scheduling algorithms and energy-aware resource management strategies have been developed to balance performance with sustainability.

2.2 Load Balancing for Live Video Analytics

2.3 Inference Serving and Resource Management in Edge Cloud Computing

As more deep learning-based applications are released as online services, managing and scheduling large-scale inference workloads in GPU datacenters has become increasingly critical. Unlike resource-intensive training workloads, inference jobs have unique characteristics and requirements that demand new scheduling solutions. The goals of inference scheduling are multifaceted, including accuracy efficiency, which can be achieved by selecting the best model for each input query and intelligently allocating resources; latency efficiency, which requires optimizing resource allocation to meet response time requirements, even for bursty and fluctuating query requests; and cost-efficiency, which involves minimizing monetary costs when using public cloud resources. These objectives are interdependent, and improving one goal may compromise another if not designed properly, highlighting the need for flexible and comprehensive scheduling systems that can balance tradeoffs between accuracy, latency, and cost.

Despite the growing need for efficient resource allocation, several solutions have been proposed to address this challenge. Clipper is a notable example of an ML inference serving system designed for real-time applications, providing support for a variety of machine learning frameworks and models, and aiming to simplify and accelerate the deployment and serving of ML models. In the other hands, TensorFlow Serving, a system developed for serving machine learning models for making predictions in real-time. It automatically adjusts to changes in traffic by adding or removing replicas. Nevertheless, none of these designs consider interference prior to deployment models share resources, instead relying on an adaptation mechanism that can lead to further performance degradation. In the other hand, Clockwork was proposed to provide predictable performance for model serving systems, by acknowledging that DNN inference has deterministic performance when running with full GPU capacity. However, their design only execute one inference at a time even when there are multiple models loaded on the GPU. In fact, Clockwork workers only overlap execution of an inference with data loading through two different CUDA Streams. Which leads to GPU underutilization as during inference all kernels that are launched would leave some resources.

Some studies have recognized that deploying multiple models on the same GPU causes interference and can lead to performance loss. For instance, others have developed a unified approach to predict latency degradation for colocated models and across a variety of platforms (CPU/GPU). This latency degradation can be used in inference serving systems to evaluate model placement. However, their approach is not fine-grained as it is based solely on model features that are the utilization of the global buffer and PCIE connection for running on the GPU device. While we recognize that these parameters play an important role, considering them does not provide a more accurate measure of their execution during inference. Another recent investigation into cloud-based inference serving highlights the challenges of model interference when multiple DNNs are run concurrently on a single GPU. The study proposes profiling to determine the optimal concurrency level, beyond which adding more DNNs reduces throughput and increases latency. Although the profiling cost is reported to be low, the method focuses solely on colocating identical DNNs, making it infeasible to profile all potential combinations where different DNNs share the same resources. The sheer number of such combinations would be overwhelming to profile.

Recent studies have explored the causes of interference in deep neural network (DNN) inference with varying levels of granularity. For instance, Abacus introduces an operator-level scheduling framework that groups operators from multiple DNNs to execute concurrently, aiming to preserve quality-of-service (QoS) guarantees. While this approach models DNNs as sequences of operators (e.g., Convolution, ReLU), it overlooks the finer granularity of GPU execution, where each operator may launch multiple kernels with distinct resource demands. Moreover, Abacus's duration model is agnostic to GPU hardware characteristics, which are critical to inference performance. Collecting sufficient profiling data across diverse GPU architectures and collocation scenarios presents a significant challenge. Additionally, the system enforces deterministic execution by waiting for all operators in a group to complete, potentially leading to underutilization of GPU resources and increased latency. Notably, Abacus operates reactively, analyzing execution only after model deployment, without offering mechanisms to assess model compatibility or predict performance degradation beforehand.

In contrast, iGnifer adopts a low-level perspective on GPU resource management by introducing an interference-sensitive inference server tailored for cloud environments. This system aims to mitigate performance degradation caused by concurrent model execution on shared GPU resources. To characterize interference, iGnifer employs several hardware-level metrics, including GPU L2 cache usage, the number of launched cores, and power consumption. While the number of launched cores is a meaningful indicator of contention, other parameters such as power and frequency are less predictive of interference severity. More influential factors—such as the configuration and scheduling of cores during kernel launches—play a critical role in shaping performance outcomes. These nuances are often overlooked in coarse-grained models, underscoring the need for more precise profiling techniques. This is acknowledged in a recent article, Usher, which proposes a kernel-level approach to interference mitigation by analyzing the achieved kernel achieved occupancy and DRAM usage during DNN inference. Usher introduces a model classification scheme that distinguishes between compute-intensive and memory-intensive workloads, recognizing that large language models (LLMs), for instance, demand significantly more memory bandwidth than computational throughput. However, from the GPU's perspective, each model ultimately translates into a set of kernels with varying resource demands, independent of the model's high-level classification.

In addition, both Usher and iGnifer rely on NVIDIA's Multi-Process Service (MPS) to enable spatial sharing of GPU resources among concurrent inference tasks. While MPS facilitates efficient resource partitioning in cloud environments, it is not supported on edge platforms such as NVIDIA Jetson. This limitation restricts the applicability of these approaches in edge computing scenarios, where hardware constraints and the absence of MPS demand alternative interference-aware scheduling strategies that operate without relying on such infrastructure.

VideoJam: Self-Balancing Architecture for Live Video Analytics 3

3.1 Background and Motivation

Introduce video analytics applications and their deployment challenges on distributed systems

Discuss the limitations of traditional load balancing approaches

3.2 Literature Review

Review existing load balancing techniques for distributed systems

Analyze their strengths and weaknesses in the context of video analytics applications

3.3 Decentralized Load Balancing Framework

Propose a decentralized load balancing framework for video analytics applications

Describe the architecture and key components of the framework

3.4 Experimental Evaluation

Present experimental results evaluating the performance of the proposed framework

Compare with existing load balancing approaches

3.5 Conclusion and Future Work (Part 1)

Summarize the main contributions of Part 1

Outline potential future research directions

Roomie: Efficient Model Cohabitation in Edge Computing Model Serving 4

4.1 Background and Motivation

4.1.1 Importance of using GPU for inference

4.1.2 Requirement for efficient DNN deployment

4.2 Literature Review

Review existing DNN deployment and optimization techniques for GPUs

Analyze their strengths and weaknesses

4.3 Kernel Understanding and Optimization

4.3.1 Kernel level scheduling

4.3.2 Architecture

4.4 Experimental Evaluation

Present experimental results evaluating the performance of the proposed approach

Compare with existing DNN deployment and optimization techniques

4.5 Conclusion and Future Work

Summarize the main contributions of Part 2

Outline potential future research directions

Appendix

Greek Letters with Pronunciations

Character	Name	Character	Name
α	alpha *AL-fuh*	ν	nu *NEW*
β	beta *BAY-tuh*	ξ, Ξ	xi *KSIGH*
γ, Γ	gamma *GAM-muh*	o	omicron *OM-uh-CRON*
δ, Δ	delta *DEL-tuh*	π, Π	pi *PIE*
ϵ	epsilon *EP-suh-lon*	ρ	rho *ROW*
ζ	zeta *ZAY-tuh*	σ, Σ	sigma *SIG-muh*
η	eta *AY-tuh*	τ	tau *TOW (as in cow)*
θ, Θ	theta *THAY-tuh*	υ, Υ	upsilon *OOP-suh-LON*
ι	iota *eye-OH-tuh*	ϕ, Φ	phi *FEE, or FI (as in hi)*
κ	kappa *KAP-uh*	χ	chi *KI (as in hi)*
λ, Λ	lambda *LAM-duh*	ψ, Ψ	psi *SIGH, or PSIGH*
μ	mu *MEW*	ω, Ω	omega *oh-MAY-guh*

Capitals shown are the ones that differ from Roman capitals.

Alphabetical Index