Distributed Computing at the Smart Device Edge

Thomas Feltin

2023

Abstract

With the increasing capacity of smart devices and an incompatibility of privacy/latency sensitive applications with cloud computing, edge computing has emerged as the best deployment solution for such workloads. In this context, this thesis studies the acceleration of heavy workloads in smart device edge networks, by providing observability through filtering of telemetry data, and a pipe-lining framework for throughput acceleration of heavy workloads. This thesis proposes a hybrid approach between cloud-out and edge-in methodologies, which leverages the multiplicity of edge compute by locally offloading computation. The thesis initially focuses on network state observability and fault diagnosis at the edge. A data-driven method to extract intelligible selections of operational features from high-dimensional network telemetry data is introduced, combining data-driven metrics and semantic information contained in meta-data, to produce selections of features which best represent an underlying event. The thesis illustrates the benefits of such a complementary meta-data analysis for data-driven fault diagnosis, highlighting the robustness of the studied approach against variations in the input feature set. With an improved understanding of the state of the edge, this thesis then studies heavy workload distribution in such environments, through the example of DNN partitioning, which consists of distributing inference workloads over several available edge devices, taking into account the edge network properties and the DNN structure, with the objective of maximizing the inference throughput. The thesis describes a process to identify partitionings which maximize the DNN inference throughput while keeping computation on the edge. The analysis of this method has lead to a set of conditions on the link between the edge network and application properties to anticipate the achieved performance and complexity, and effectively size an edge network environment. Finally, the thesis describes a dynamic partitioning framework to improve the system performance and robustness, which leverages the observability of the network to adapt to heterogeneous and dynamic edge networks.

Keywords — Distributed Computing, Edge Computing

Résumé

Avec la capacité croissante des appareils intelligents et une incompatibilité des applications sensibles la latence et la préservation de la vie privée avec le cloud computing, l'informatique en périphérie (edge computing) est devenue la meilleure solution de déploiement pour ce genre de charges de travail. Dans ce contexte, cette thèse étudie l'accélération d'importantes charges de travail dans les réseaux de périphérie d'appareils intelligents, en fournissant une observabilité par filtrage des données de télémétrie, ainsi qu'un cadre de pipelinage pour l'accélération du débit de ces charges. Cette thèse propose une approche hybride entre les méthodologies cloud-out et edge-in, en exploitant la multiplicité des appareils en périphérie afin de décharger localement le calcul. La thèse se concentre initialement sur l'observabilité de l'état du réseau et le diagnostic des pannes en périphérie. Cette thèse présente une méthode de sélection sémantique de caractéristiques opérationnelles à partir de données de télémétrie de réseau à haute dimension, combinant des métriques axées sur les données et des informations sémantiques contenues dans les métadonnées, afin de filtrer des des caractéristiques représentant au mieux un événement sous-jacent. La thèse illustre les avantages d'une telle analyse complémentaire de métadonnées dans le diagnostic de pannes, mettant en évidence la robustesse de l'approche étudiée face aux variations des caractéristiques d'entrée. Avec une meilleure compréhension de l'état du réseau de périphérie, cette thèse étudie ensuite la distribution des charges de travail lourdes dans de tels environnements, à travers l'exemple du partitionnement de réseaux de neurones profonds, qui consiste à distribuer ces travaux d'inférence sur plusieurs appareils disponibles, en prenant en compte à la fois les propriétés du réseau et la structure du réseau de neurone, dans le but de maximiser le débit d'inférence. La thèse décrit un processus pour identifier les partitions qui maximisent le débit d'inférence, en maintenant le calcul localement. L'analyse de cette méthode conduit à un ensemble de conditions sur le lien entre le réseau de périphérie et les propriétés de l'application pour en anticiper les performances et la complexité, et dimensionner efficacement un environnement de réseau en périphérie. Enfin, la thèse décrit un cadre de partitionnement dynamique pour améliorer les performances et la robustesse du système, qui tire parti de l'observabilité du réseau pour s'adapter à ces réseaux hétérogènes et dynamiques.

Mots-clefs — Calcul distribué, informatique en périphérie de réseau

Remerciements

Remerciements.

Contents

Part I

Introduction

Chapter 1

Introduction

In the 2020s, the availability of computing and networking technologies, along with their decreasing cost, has made a commodity of smart devices, *i.e.*, electronic devices that are connected to the Internet and can communicate with each other. These devices range from smart home appliances, *e.g.*, thermostats and security systems, to wearable technology, *e.g.*, fitness trackers and smartwatches. Smart devices can make daily tasks more convenient and efficient, *e.g.*, a smart thermostat can learn a user's schedule and adjust the temperature of a home accordingly, saving energy and money, and they also have the potential to improve safety and security by providing real-time monitoring and alerts. Smart devices close to end users are found in homes, in healthcare, networking, in smart cities, in industry, in energy, or in agriculture. Across all these applications, connected devices have continuously generated an estimated 2.5 quintillion bytes of data every day in 2020[1]. Storing, transporting, and processing this information have been among the principal factors behind evolutions of distributed computing. In the early days of computing, when device capacity was insufficient to consume data close to its source, mainframe computing was the norm, *i.e.*, a large computer received instructions from smaller terminals. In the 1980s, with the advent of personal computers, came the popularization of software which could be run entirely on devices close to users. In the 2010s, with the rise of cloud computing, the computation moved "back" to some distant data center. This, as the size of the workloads increased and needed more resources to compute.

These incremental changes in the location of data processing are also linked to emerging trends in technology. Different types of workloads, *e.g.*, Artificial Intelligence, Internet of Things, or Virtual Network Functions, each have a different profile of transmitted data volumes and processing complexity, which match a specific deployment paradigm. For example, Artificial Intelligence usually requires a lot of computing capacity to run with dedicated hardware, but has little requirements on the network, while Virtual Network Functions are less computing resource intensive but require network bandwidth, by nature. Because of these profiles, in 2019, 96% of Artificial Intelligence tasks were computed in the cloud[2], because it allowed developers to deploy with large controllable capacities, while Virtual Network Functions

[1] According to the *Data Never Sleeps* annual study from Domo https://www.domo.com/learn/infographic/data-never-sleeps-8

[2] According to a survey from Nucleus Research Inc., 96% of Deep Learning based applications ran in the cloud in 2019. https://d1.awsstatic.com/whitepapers/Deep%20learning%20on%20AWS.pdf

such as firewalls or encryption were kept on premise. Every technology trend can be mapped to a deployment model that fits its computational needs and maximizes its performance.

External requirements such as privacy considerations can restrain workloads from being run in their highest performing location. For example, self-driving cars heavily rely on Artificial Intelligence to analyze their environment and take decisions. Response times of autonomous vehicles are required to remain under 100 ms [1] to ensure road safety. However, because of the mobility of autonomous vehicles, and their geographical distance from cloud data centers, the data acquired by the vehicle can not be exported for computation, the communication latency would exceed the 100 ms response time constraint. Consequently, although these workloads require capacities mainly found in data centers, the nature of autonomous driving is incompatible with cloud computing. Additionally, the collected data can be sensitive and protected by privacy policies, *e.g.*, the General Data Protection Regulation in the European Union (GDPR) [2]. Exporting private data to a public cloud for processing is, in many cases, prohibited by privacy considerations, forcing the processing to be located close to the data source, in a secure environment. More generally, some technologies may have strong links with a specific computing paradigm, but properties of the data (*e.g.*, privacy restrictions) or of the expected performance (*e.g.*, low-latency/high-bandwidth applications) can restrain workloads from accessing the needed resources. In the examples above, the high latency of cloud computing and privacy considerations restrain these workloads from running in the cloud.

As a result, the edge computing paradigm has emerged, and captures use-cases in which computation is required to stay close to data sources, and the available resource do not match the required capacity.

The overarching question explored in this thesis is whether, and how, it is possible to provide tools for deploying and monitoring resource-intensive applications at the edge while preserving the convenience and performance of cloud deployments.

The remainder of this chapter is organised as follows: Section 1.1 presents background on distributed computing and the evolution in platform and applications which have led to developments of the cloud and edge computing paradigms. Section 1.2 presents the existing technologies and mechanisms which enable computation at the edge, by distinguishing two main approaches, respectively derived from the cloud and the IoT. Finally, section 1.3 argues for the description of a hybrid approach, which leverages mechanisms from both methods to exploit idle smart device resources and enhance the performance of latency and privacy sensitive workloads. This chapter concludes by providing general assumptions and notations which will be used throughout this thesis in section 1.4.

1.1 Background

This section introduces background on distributed computing systems, with evolutions in platforms and applications, leading to the definition of edge computing. Section 1.1.1 presents the history of computing platforms from the 1960s to the adoption of cloud computing. Section 1.1.2 presents implications of these platform changes in the design and deployment of applications over time. Section 1.1.3 out-

lines the definition and advantages of edge computing by depicting its dual origin in both cloud computing and IoT. Finally, section 1.1.4 presents properties of the resulting distributed systems and their advantages.

1.1.1 From mainframe to cloud computing

In the 1960s, with mainframe computing, computing units provided time-sharing services to clients interacting via local "dumb" terminals which allows only data entry and its visualization. Mainframes were entirely deployed and maintained in each organization, to deliver domain-specific services.

With the development of digital networks in the late 1960s and early 1970s, clusters of interconnected computing units showed similar performance to mainframes and supercomputers, leading the way for new resource abstractions and programming frameworks. The idea that higher computing performances could be achieved by interconnecting off-the-shelf computing elements led to the creation of ARPANET in 1969, and later the TCP/IP communication protocol which standardized communication between independent networks.

In the 1980s, the development of the Internet, facilitated by TCP/IP and the formalization of the Domain Name System (DNS), which facilitates the addressing of connected devices, gave rise to the emergence of personal computers (PCs). This period led to the creation of the World Wide Web, which eventually allowed applications to be accessible from anywhere in the world, further abstracting application deployment localisation, and allowing use-cases to transition from specialized research applications to the broader public, with services accessible to everyone from home.

In the 1990s, content delivery networks (CDNs) emerged, as a consequence of the large demands in Web services causing congestion and bottlenecks when serving large volumes of data over long distances [3]. Efficient data delivery was achieved by adding caching proxies, *i.e.*, servers which store the frequently or most recently requested content close to the requests. CDNs rely on the structure of their network and the strategic placement of these caching proxies to ensure good quality of service to the end users. This mechanism lowers the request latency for the end user, diminishes the bandwidth consumption, while lowering the computational load on the centralized server [4]. These systems were among the first standardized distributed systems used to enhance service performance.

The concept of grid computing emerged around the same time, in the late 1990s. Grid computing consists of sharing resources in a large-scale network environment to perform tasks which would otherwise take significantly more time on a single machine [5]. This distributed computing paradigm consists in connecting remote and diverse computers together to jointly perform a resource intensive task. This implies integrating open standards and collaboration mechanisms between devices in order to offer services which share large pools of resources across a large number of clients. In opposition to supercomputers, which interconnect processors through high speed connectors to transmit computation results, grid computing workloads are highly parallelizable and are processed on individual and complete computers. In parallel, with the development of Napster, a music file sharing network, came the development of peer-to-peer overlay networks. Peer-to-peer networks are defined as virtual networks in which nodes share a part of their own resource (content, storage,

or computing cycles), and are simultaneously consumer and producer of these resources [6]. Such networks leverage the same connectivity as grid computing, but all nodes (or peers) have identical roles, and interact directly between each other, rather than through a centralized server or authority. In addition to enabling shared and increased computing capacity, peer-to-peer networks allowed for the implementation of properties such as fault-tolerance, the adaptability to dynamic network topologies, object location and load balancing. What grid computing and peer-to-peer computing have illustrated is the ability to reach a computing capacity comparable with supercomputers, based solely on interconnected commodity hardware, at a much lower cost.

In the 2000s, developments in virtualization, cluster computing and middleware resulted in the creation of what has become known as "cloud computing". Cloud computing is a model for on-demand access to a pool of shared resources that support virtualization and can be easily provisioned, along with development platforms and services. Cloud computing allows applications to be deployed and scaled without the hardship of managing the underlying infrastructure [7]. This model allows users to benefit from powerful hardware and scale their infrastructure needs quickly and on-demand. There are several types of cloud: public clouds, for which all software, hardware, and infrastructure is operated by cloud vendors and shared with other users (*e.g.*, Amazon Web Service, Google Cloud Platform, Microsoft Azure) private clouds, in which the infrastructure is reserved for a single organization and services and applications are kept on a private network, and hybrid clouds which allow applications to use ressources simultaneously from private and public cloud infrastructures.

1.1.2 Evolutions in application structure

The evolutions in computing platforms and paradigms described in section 1.1.1 have also impacted the way applications were designed.

With mainframe computing, applications were only required to run on a single computer, often located in an organization, with domain-specific knowledge kept in a single location. Applications were monolithic, *i.e.*, all components were combined into a single program, and for a single platform. With the adoption of personal computers, users became able to design their own applications, further driving development of software engineering. Personal computers also caused users to handle and store data for the first time, with the implied security considerations. This created a large scale development of *single-tiered* applications, which are not required to interact with a remote entity to deliver a service, and are either installed on a personal computer or on-premise, *e.g.*, the Microsoft Office suite, or simple data stores.

The development of the Internet, and later the World Wide Web, changed application structures. Instead of being single-tier, applications initially became composed of two components: a light-weight application on the user side, communicating with a heavier server-side application, which handles the business logic. Standardization led to the definition of a client-server topology, with organizations deploying and maintaining their own infrastructure to support growing request volumes from clients. The user-side application mostly consists of a user interface (UI), *i.e.*, a graphical representation of the offered service for users to interact with, which com-

municates through an application programmable interface (API), *i.e.*, a standard communications protocol with the server, which handles all the business logic. The advantage of such a topology is to keep the client side application as light-weight as possible to improve performance and to facilitate testing and maintenance.

Increasing demands have led applications to become more and more distributed, with organizations growing towards higher geographical distribution of sites. This implies that an organization could require access to common resources among sites. This distribution, along with the ability to access anything anywhere, accelerated the definition of new application topologies, *e.g.*, the 3-tier architecture, with the server-side application separated into a back-end application and a data store.

Further modularity in applications have extended the above principles to the definition of *N-tier applications*, and Service Oriented Architectures (SOAs) in which application architectures were defined based on application logic, rather than platform constraints. This allows developers to develop, test, and maintain individual components individually, with the World Wide Web allowing placement of each component in any of the organization data-center, all around the world.

Even with N-tier applications, managing and provisioning the infrastructure requires continuous efforts, with continuous updates and security patches. Furthermore, administrators are required to dimension their hardware in order to handle peak load on an applications, which implies that most of the hardware is idle for the majority of the application life cycle.

With virtualization and cloud computing, applications moved to third-party cloud providers, where resources could be allocated and shared dynamically, allowing applications to only pay for the necessary hardware, while out-sourcing the infrastructure maintenance. Containerization allowed applications to be even more compartmentalized, abstracting everything below the application from developers, *e.g.*, infrastructure, operating system, security, and networking. As a result, modern applications are compounds of micro-services, *i.e.*, compact and portable instances of standardized programs which can be easily deployed anywhere. This convenience for developers has led to a wide adoption of the cloud, where applications could be easily deployed to hardware.

1.1.3 The smart device edge

Edge computing is a distributed computing paradigm that brings computation and data storage closer to the edge of the network, where devices and end users are located. This approach has gained popularity in the 2020s due to the proliferation of Internet-connected devices and the emergence of applications that require low latency and privacy. However, this definition can cover different contexts, as there are different types of edge networks, as depicted in figure 1.1. To better understand the range of edge computing applications, it is important to know where the concept originated. With this in mind, there are two ways to think of the edge, referenced as *cloud-out* and *edge-in*.

Cloud-out

The *cloud-out* approach refers to the edge as *a computing paradigm which relocates computation tasks from data centers to the close proximity of data sources, while keeping the experience of the cloud.* Cloud computing has proven to be a

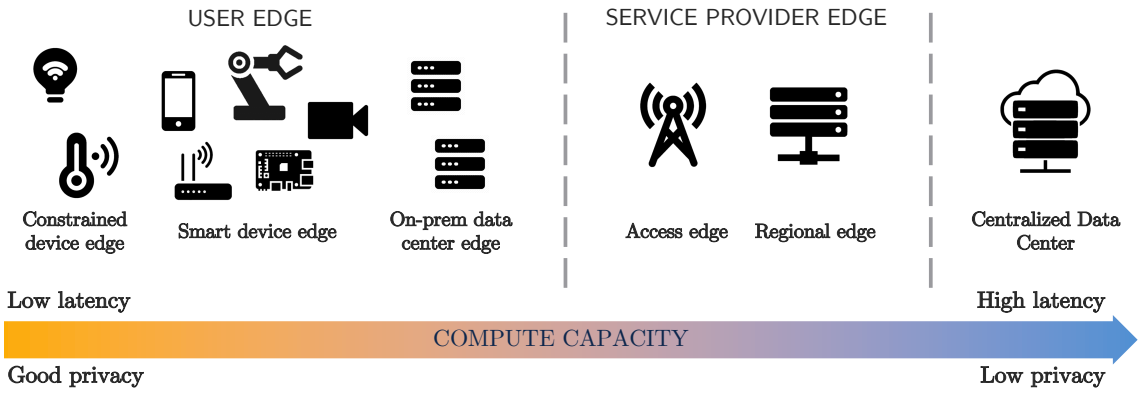


Figure 1.1: The edge computing continuum, from constrained devices to centralized data centers [8].

popular medium for deploying applications, due to the availability of large controllable computational resources, without the hardship of infrastructure management. With computing, storage, and network management functions off-loaded to cloud providers, application developers found, in cloud computing, an easy solution for fast application deployment. However, the early 2020s have shown that more and more developers are choosing to bring computation back on premise, close to data sources [3]. The reason for this migration is two fold. Firstly, depending on the locality of an application, using public clouds can induce important latencies, caused by the wide area network (WAN), or simply the geographical distance to the cloud network. Cloud computing solutions can also be expensive, with important monetary costs of computation and storage in a public cloud. Secondly, in the 2020s, use-cases with incompatible requirements with the cloud have become prevalent. In addition to latency restrictions, privacy policies can restrict the data and computation from leaving a proprietary or secure network. In such cases, privacy considerations make it impossible for some applications to run in the cloud. However, the cloud computing paradigm still attractive for its convenience in terms of deployment, continuous development and integration, the *cloud-out* vision of the edge focuses on maintaining the same convenience, while keeping data and computation locally.

Edge-in

The *edge-in* vision considers the edge as *an up-scaling of computation capacities of devices close to data sources, to keep computation and storage close to devices and users*. The edge can be defined from the evolution of constrained devices and the Internet of Things (IoT). With the proliferation of connected devices and the popularization of 4G/5G networking, IoT has created a new means of consuming and processing data. Advances in systems on a chip have allowed operating systems to be embedded on constrained devices, and to run heavier computation close to data sources. Standard IoT devices are defined by two elements: a *sensor* and an *ability to connect to the Internet* to transmit sensor data. With increasing computation capabilities, devices have gained the ability to run workloads in the close proximity of

[3]Gartner Infographic: Understanding Edge Computing, https://www.gartner.com/en/doc/750789-infographic-understanding-edge-computing

the sensor, *i.e.*, on the same edge device. As IoT and other applications that require low latency and high bandwidth have become more prevalent, edge computing was developed as a way to address these device capacity limitations by bringing computation and data storage closer to the edge of the network. This reduces the need for data to be transmitted over long distances, improving the speed and reliability of data processing.

Whether the edge is defined from its origin in the cloud or in IoT, the reality it embodies is somewhere in this middle ground, covering many different settings. There are different edges depending on two factors, *i.e.*, the closeness to the network edge and the computation capacity. Figure 1.1 [8] depicts the edge computing continuum, displaying the different edges from closest to the network edge with the least computing capacity (left), *i.e.*, IoT or constrained device edge, to furthest from the edge with most computing capacity (right), *i.e.*, data centers in the regional edge.

No matter the context, when modeling the edge, there are two main invariants that appear in each of the categories in the edge continuum of Figure 1.1.

- *Locality*: The main defining characteristic of the edge, is its proximity to the edge of the network, where devices and users are located.

- *Data source*: The edge is located close to data sources. Edge applications focus on enhancing the computing capacity of sensor networks to process information on their environment.

- *Low latency and privacy* As a consequence, the edge serves applications which need to stay close to data sources and users, *i.e.*, low latency and privacy constrained applications such as video streaming, augmented reality, or artificial intelligence.

The remainder of this work focuses on the smart device edge. This category corresponds to devices located outside of centralized data centers, close to the data sources, but with some ability to contain heavier workloads. Hardware in this category can support virtualization or containerization, as well as cloud-native applications, *i.e.*, applications taking advantage of the distributed computing capacity of the cloud. Such devices include consumer mobile devices, *e.g.*, smart phones or PCs, gateways and servers for IoT applications, or connected objects with efficient compute, *e.g.*, smart cameras. While powerful enough to run general workloads, smart devices are still much more constrained than servers available in data centers.

1.1.4 The benefits of distributed computing

The term *distributed computing* can refer to a variety of different approaches, including the use of multiple computers connected by a network to work on a single problem, the use of a grid of computers to perform independent tasks in parallel, or the use of distributed systems to manage and coordinate the actions of multiple devices. Overall, a distributed system can be defined as a group of independent entities interacting with each other to achieve a common goal [9]. With distribution enabling complex application structures, designing distributed systems consists of

identifying service-level properties, *e.g.*, redundancy, load-balancing, fault-tolerance, etc, which are enhanced by a given distributed application architecture.

When modeling distributed computing systems, the individual entities are referred to as *compute nodes* and a *computation* or a *run* is the execution of a workload across these compute nodes. The basic assumptions that define a distributed computing system are (i) the set of compute nodes appears as a single system to the end user, (ii) the individual compute nodes do not share a common memory or a common clock, and (iii) the individual compute nodes communicate with each other via message passing through a communications network.

Examples of distributed computing systems include Apache Spark [10], based on the MapReduce framework [11], or the bitcoin network [12]. There are several common topologies, or types of interactions between compute nodes, *e.g.*, the client-server relationship consists of clients sending instructions to a server concentrating the majority of the computation and data, the multi-tier relationship separates parts of the compute to different compute nodes according to functional separations, or the peer-to-peer relationship considers all nodes to be equally responsible for the management and computation of workloads. Depending on the architecture, distributed computing systems can be designed to offer different advantages, when compared to single node systems:

- *Sharing of data and resources*: One of the main benefits of distributed systems is their ability to handle large amounts of data and computation efficiently. By dividing a problem into smaller parts and distributing it across multiple devices or computers, a distributed system can process data in parallel. Some applications are also geographically distributed by nature, *e.g.*, with data located in different locations and unable to be replicated at every compute node.

- *Robustness*: Due to their distributed nature, distributed systems can be designed to be less vulnerable to single points of failure. If one device or computer fails in a distributed system, provided the system is designed to detect and exclude the failing device, can continue to function, ensuring that the overall system remains operational. Robustness includes *availability*, *i.e.*, resources should be available at any time when requested, and *partition-tolerance*, *i.e.*, the system should be able to function with individual compute node failures or loss of connectivity.

- *Scalability*: As the needs of an application change, a distributed system can easily add or remove devices to meet the demand for computation. This allows a distributed system to adapt to changing conditions and grow or shrink as needed.

- *Modularity or flexibility*: Separating functions to individual compute nodes allows components of the system to be modified and updated without impacting the overall performance.

These characteristics and design decisions heavily rely on properties of the data, *i.e.*, volume, type, locality, or privacy considerations. The increasing production and processing demands of data is often described by the four Vs: volume, *i.e.*, large quantities of data and workloads, velocity, *i.e.*, data and workloads are generated at

great speed, variety, *i.e.*, heterogeneity in the type of data produced, and veracity, *i.e.*, data can have different degrees of truthfulness or relevance in this large influx.

In this context, distributed computing over smart devices at the edge of a network offers a number of benefits. For example, edge devices can process data locally, reducing the amount of data that needs to be transmitted over the network and improving the speed and efficiency of the overall system. Edge devices can also operate independently, allowing them to continue functioning even if the rest of the network is unavailable or offline. In addition, edge devices can be used to perform tasks that require low latency, such as real-time control of industrial processes or self-driving vehicles.

1.2 Distributed computing at the edge

This section presents related work on computing paradigms at the edge. As described in section 1.1.3, there are two main strategies for bringing compute capacity in edge networks. Section 1.2.1 describes related work on computation offloading of workloads to higher capacity environments (cloud-out approach), and section 1.2.2 presents solutions embedding compute close to the edge (edge-in approach).

1.2.1 Cloud-out: Computation offloading at the edge

There are three tiers of computing resources considered, when modeling the edge: (i) the constrained devices, *i.e.*, devices producing data with limited processing capacity, (ii) the edge computing, *i.e.*, some computing device located close to data sources, *e.g.*, a smart device or a dedicated server, and (iii) the cloud. From these three tiers, there are two general architectures when considering computation offloading at the edge: the two-tier architecture, which consists of constrained devices and edge computing (better suited for latency or privacy-sensitive applications since workloads are processed at the edge), and the three-tier architecture, which consists of constrained devices, edge computing, and cloud computing (better suited for modeling performance optimization and application management) [13]. The main models presented in this section are cloudlets, and Mobile-Edge Computing (MEC).

Cloudlets [14] seek to solve the latency problem when accessing the cloud, by adding cloud-like computing resources at the edge of the network for task offloading. To support heterogeneous workloads, cloudlets support virtualization and containerization to allow computation offloading of low footprint virtual machines and containers, allowing cloudlets to only store soft states, and lower the impact of the computation. Cloudlets are decentralized and widely dispersed Internet infrastructure components which are made available to edge devices, in this two-tier architecture.

Mobile-Edge Computing (MEC) provides cloud computing capabilities within a Radio Access Network (RAN). MEC contributions consist of partitioning and/or offloading computation tasks. MEC computing systems typically consist of three components: (i) a task partitioning mechanism, *i.e.*, a partitioning policy defining whether tasks can be partitioned and in what manner, (ii) an offloading decision or task placement, *i.e.*, deciding which device will receive the whole or partitioned task to process, *e.g.*, locally, in an edge server, or in the cloud, and (iii) resource allocation, *i.e.*, determining the amount of resources allocated, *e.g.*, computing, communication, and energy. Offloading can be done in a centralized or decentralized manner, with

either a global or local view in the decision process. Depending on the number of edge devices and offloading capabilities involved (edge or cloud), offloading scenarios can be one-to-one, one-to-many, many-to-one, or many-to-many.

Computation offloading is often defined as an optimal placment problem, with a defined optimization objective, *e.g.*, latency minimization, energy consumption minimization, drop rate minimization, throughput maximization, computation efficiency maximization, or monetary cost minimization. The offloading problem can be defined based on one or a combination of these metrics. There are three main technologies for partition offloading and scheduling: (i) Convex, non-convex and Lyapunov optimization processes, (ii) Markov Decision Process (MDP) and Reinforcement Learning (RL), and (iii) game theory.

Related Work

Gabriel [15], a system for wearable cognitive assistance, uses a cloudlet architecture to support intensive low-latency computation. Similarly, Content Delivery Networks or Web service can benefit from addition of cloudlet capacities [16], by optimizing computing load on edge devices.

Within MEC, with a fixed set of resources, related work focuses on defining an optimization problem that optimizes certain oobjectives. MULTIUSER MECO and EPCO [17,18] minimize energy consumption of EDs, by defining models for computation and transmission of data, and assuming that nearby edge servers are available with sufficient capacity. Both methods use Karush–Kuhn–Tucker (KKT) conditions to solve the optimization problem. SDTO [19, 20] minimizes latency by defining a Mixed Integer Nonlinear Programming (MINLP) problem and solving it as a 0-1 integer programming problem. HGPCA [21] uses more parameters which makes the problem unsolvable, and they therefore use a combination of two heuristics. The throughput problem is modeled in [22,23] and solved by using the Alternating Direction Method of Multipliers (ADMM) method to separate and solve the optimization problem. In fog computing use-cases with three-tier architectures, another ADMM-based method is presented in [24], in a context with EC servers offloading sub-tasks to the cloud while preserving privacy. Multi-factor objective functions, *e.g.*, joining latency and energy consumption, are considered in [25] which uses the Interior Point Method (IPM) to solve the optimization problem, or [26] which relaxes the NP-hard problem by using the Semi Definite Relaxation (SDR) method to find a near optimal solution. All previous optimization methods can be inconvenient at the edge, since they require a powerful centralized node to run the optimization. [27–29] use Lyapunov optimization to model one-to-one or many-to-one offloading from energy harvesting devices to an edge server, with the objective of minimizing both the latency and dropping of workloads. Lyapunov optimization doesn't support time-dependant variables but the weighted perturbation method can be used for this purpose. [30] extends the offloading modeling by including the edge server's queue in the optimization process. [31] focuses on energy harvesting in offloading for multi-user, multi-task MEC. However Lyapunov optimization (i) doesn't reach an optimal solution, and (ii) assumes time independence in actions, which usually is not the case, *e.g.*, for energy queues.

Using Markov Decision Processes (MDP) can allow each agent to make a placement or offloading decision for itself, which enables a decentralized system. For example, ST-CODA [32] defines a MDP in an three-tier environment to model the

processing time and energy consumption of different computation nodes, as a function of the transmission cost in heterogeneous networks. Similarly, the MDPs defined in [33, 34] are used to minimize edge device energy consumption while satisfying a delay requirement and use the Post-Decision State (PDS) method to solve the problem in real-time. The curse of dimensionality can quickly become an issue when modeling offloading problems with MDPs. Therefore, some research work [35–38] uses Q-learning [39] to simultaneously optimize metrics such as latency and energy consumption, when the dimensionality of the problem is too high or when the statistical distributions of the metrics are unknown due to the nature of the edge devices, *e.g.*, energy harvesting or highly mobile edge devices. The use of DNNs to solve the Reinforcement Learning (RL) problem can also help deal with the high complexity. For example, deep RL can be used to model the complexity of the Q-table and make decisions on offloading to road side units in [40], or to aerial vehicles in [41], both acting as edge compute. Game theory can also allow for decentralized decision, *e.g.*, [42] describes a potential game, defining a potential function and searching for the Nash equilibrium with several methods such as Fully Distributed Computation Offloading (FDCO) [43]. [44] defines the problem as a General Nash Equilibrium Problem (GNEP), *i.e.*, with edge devices strategies depending on other devices.

1.2.2 Edge-in: Enhancing connected device capacity

The principal enabler of heavy workload computation at the edge is the evolution of the available computing capacity of edge devices. Processor capacity has increased from a few MHz in the 1980s, *e.g.*, with the Intel 8080 processor reaching a processing speed between 2MHz and 3MHz, to several GHz in 2020, *e.g.*, with the ARM Cortex-A72 processing unit used on a Raspberry Pi 4, containing four 1.5GHz cores. Development of systems on a chip (SoC) were driven by increased demand in cell phones in the 1990s, and IoT in the early 2000s. Commonly used SoCs in 2020, such as the Raspberry Pi or the NVIDIA Jetson Nano, contain general-purpose operating systems, with CPUs, GPUs, memory, and input/output management. In the transition from IoT to edge, limited, power-efficient, and application-specific hardware has become more and more general-purpose, with higher computing capacities.

However, several technological trends also revolve around leveraging the available hardware to perform heavy computation locally. One of the earlier instances of such a technology is described in 1997, with pervasive or ubiquitous computing, introduced as a paradigm which takes advantage of the multiplicity and connectivity of devices with available computing capabilities to create spaces where the computation physically moves with the user [45]. Ubiquitous computing revolves around the design of smart spaces which integrate computing capabilities in infrastructure, *e.g.*, in meeting rooms, corridors, etc. The key concept of ubiquitous computing is invisibility, *i.e.*, the users become unaware of the technology around them. *"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it."* [46]. This paradigm was one of the first to study computation embedded in devices close to end users. The field of ubiquitous computing mainly tackled issues surrounding (i) scalability, with the smart spaces and number of users growing, and leading to exploding complexity in the interactions between personal computing spaces [47], and (ii) heterogeneity, with protocols needing to mask the differences in the "smartness" of different de-

vices, to make mobility seamless. The concept of ubiquitous computing and the progressing development of information technology, have led to the description of the Internet of Things (IoT). This field of study focuses on the interconnectivity of sensors and devices which continuously collect and exchange data.

Ubiquitous computing and IoT are related to edge computing, since these concepts pursue similar use-cases. The main differences lie in the point of focus of each of these fields of study [48]: while ubiquitous computing focuses on invisibility of the computing capacity embedded in physical objects, IoT rather focuses on connectivity, and edge computing on the performance enhancement of these environments. Ubiquitous computing and IoT have allowed to move the network edge closer to end users. As a result, edge computing brings higher compute capacity to this extended network edge, when the latency constraint of the WAN, and the constrained resources of IoT/embedded devices, cannot respond to the increasing application requirements.

The concept of fog computing was introduced in 2012, and is designed for scalability, *i.e.*, to handle a large number of IoT devices and big data volumes for real-time low-latency applications. Fog computing is a paradigm that models computing resources between end devices and the cloud. According to the OpenFog reference definition [49,50], *fog architectures selectively move compute, storage, communication, control, and decision making closer to the network edge where data is being generated in order to solve the limitations in current infrastructure to enable mission-critical, data-dense use case.* Fog computing contains both cloud-out and edge-in approaches, since it considers placement of edge servers close to data sources, *i.e.*, dedicated servers designed to receive offloaded computation tasks, and methods which leverage otherwise useful devices (access points, routers, switches, etc.). Resources made available at the edge in the context of fog computing are called fog nodes, and can range from low capacity devices such as sensors, connected devices, access points or routers, to higher capacity devices such as cloudlets [51].

Related Work

Computing capacities of systems on a chip (SoC) illustrate evolutions in edge network capacity. Early small-sized processors were originally used in calculators and electronic watches. In 1972, when Intel introduced the Intel 8008 8-bit CPU, with a processing rate of approximately 500 kHz, followed by the Intel 8080, released in 1974, reaching 2 to 3 MHz. Other 8-bit processors, released in the 1970s, with comparable processing rates, include the Motorola 6800, the MOS 6502, the Zilog Z80, or the Intel 8086, reaching a maximum clock rate or around 10 MHz. These processors were later included in the first instances of personal home computers, such as the Apple II, the Commodore PET, or TRS-80, which were 8-bit personal computers including MHz processors, and 4kB or memory. Around the same time, in the late 1970s, came the first SoCs which could also include input/output management, storage, and a graphical processing unit. In 2022, the Raspberry Pi 4, a common single-board computer used in edge computing applications, includes four 1.5GHz CPU cores, with 2 to 8 GB of memory. This evolution in capacity has allowed such devices to support general-purpose operating systems and support a variety of applications and virtualization. Specialized hardware such as the NVIDIA Jetson graphical processing units, or the Google Coral tensor processing units are included in such systems in support of developments in Artificial Intelligence at the

edge.

Ubiquitous computing efforts include projects such as Aura (Carnegie Mellon University), Endeavour (University of California, Berkeley), Oxygen (Massachusetts Institute of Technology), Portalano (University of Washington), and industrial attempts at AT&T Research in Cambridge, and at the IBM T.J. Watson research center. These research efforts consist of embedding computing capabilities in every day objects, and making mobility of workloads invisible to the end users.

The fog computing paradigm is introduced in 2012, and its most significant contribution is the IOx network infrastructure products[4], which is a framework for making computing capacaity available on network infrastructure, mainly to support IoT applications. Applications of fog computing include real-time video analytics, virtual or augmented reality (VR/AR), autonomous driving, and networking solutions such as network function virtualization (NFV) [51].

1.3 Thesis statement: Distributed computing at the smart device edge

The evolution of distributed computing platforms and applications described in sections 1.1.1 and 1.1.2 have depicted changes in the locality of computing driven by performance, with applications requiring more and more resources over time. Section 1.1.4 has described the many other advantages of distributed systems, which implies that all these properties have been secondary since the 1960s, and that placement was always chosen based on available computing capacity and convenience. With this observation, section 1.1.3 has described edge computing, a computing platform development driven by latency and privacy motivations, and not availability of computing resources. Then, section 1.2 has introduced two main concepts for enabling computation at the edge, with instances of cloud-out approaches, which consist of processes for offloading workloads to nearby servers, and edge-in approaches, which consist of increasing the computing capacity of edge devices to leverage the available computing resource.

With these methodologies described, the smart device edge can also be, in itself, a good candidate for hosting heavy workload computation at the edge.

- In 2022, general-purpose smart devices with GHz capacity and specialized hardware are a commodity (*e.g.*, Raspberry Pi or NVIDIA Jetson Nano), making the smart device edge a good candidate platform, for heavy latency and privacy sensitive applications.

- High degrees of connectivity and available bandwidth in smart device edge networks (*e.g.*, through 4G/5G cellular networks) facilitates communications among smart devices in edge networking contexts, enabling pooling of smart device resources to increase individual device capacity.

- Cloud-out approaches, which enable heavy workload computation by placing nearby computing elements to receive offloaded tasks, requires continuous

[4] https://www.cisco.com/c/en/us/products/cloud-systems-management/iox/index.html

deployment and maintenance of computing capacity in order to meet. Furthermore, increased mobility complicates cloudlet-type applications, or sizing of specialized hardware such as fog nodes, since demands at the edge are challenging to predict.

These arguments favor hybrid deployments at the edge, with a combination of cloud-out and edge-in approaches, to favor the exploitation of idle smart device resources across the edge network to increase the overall computation capacity.

Pooling resources together in a smart device edge environment implies, similarly to distributed computing environments, to have a sufficient understanding of the state of the system, and of its individual parts, and to find orchestration mechanisms for the computation. However, the fact that these distributed systems run on the smart device edge involves specific constraints. In addition to being limited in compute compared to edge servers or dedicated compute, smart devices have, by definition, a primary workload continuously running (*e.g.*, forwarding packets for routers, recording for smart cameras, etc.), with varying capacity left for secondary tasks. Similarly, edge environments imply heterogeneity in connectivity and bandwidth between smart devices, which needs to be taken into account in the modeling. Finally, this thesis focuses on workloads which are too heavy to be run on a single device, favoring distribution on the edge.

Therefore, the different steps in the design of distributed systems for the smart device edge, which are explored throughout this thesis, are:

- identification of the limiting resources in a smart device edge between bandwidth and processing capacity, and extraction of their states in a constrained environment (part II).
- data-driven detection and explanation of events or faults occurring on the smart device edge network, without knowledge of the event profile and behaviour (part II).
- data-driven detection and explanation of events or faults occurring on the smart device processors and computing capabilities (part II).
- modeling and profiling of heavy workloads to run on the smart device edge, to learn a computation profile with satisfactory levels of precision (part III).
- definition of a partitioning and workload placement scheme to leverage the smart device edge according to a pre-defined objective (part III).
- identification of deterministic conditions on the smart device edge network properties leading to the possibility of performance improvement by resource pooling (part III).
- dynamic monitoring of telemetry and state adjustment to keep optimizing the performance and adapting to state changes in the system (part III).

In sum, such a distributed system on the "smart device edge" offers observability over the network and device state (part II), event detection and explanation (part II), and dynamic application profiling and distribution (part III), to fully leverage the computing capability of the edge.

As will be studied throughout this manuscript, these systems show benefits in processing capability and observability, while preserving latency and privacy requirements of applications.

1.4 Definitions and working assumptions

This section introduces the working assumptions that will be used throughout the manuscript, and defines notations which will be used recurrently.

- A *smart device* is a computing device, capable of performing data processing and analysis at or near the source of the data. Such devices are often equipped with sensors, processors, and storage capabilities, and they can be programmed to perform a wide range of tasks, such as data filtering, aggregation, and analysis. Examples of smart devices at the edge include security cameras, networking equipment, and wearable devices that can perform data processing and analysis locally. The principal assumption for the remainder of this manuscript is that smart devices have autonomy, connectivity, context-awareness, and a purpose-built task.

- A *feature* describes a measurable property or characteristic of a monitored object, *i.e.*, the columns of the telemetry dataset. For example, in a dataset containing medical records, features might include a person's name, gender, age, height, weight, etc. Each of these features provides a different piece of information about the presented individual. Features can be categorical or numerical. Categorical features take on values from a finite set of categories or labels, while numerical features take on numeric values.

- A *time-series* is a set of data points that are collected at regular intervals over time. Time-series data is commonly used to analyze trends, forecast future events, and understand patterns over time. In a time-series, the data is organized into a sequence of observations, where each observation corresponds to a specific point in time. Time-series data can be univariate or multivariate, depending on whether they consist of a one or multiple observed features over time.

Finally, generic mathematical notations used throughout this thesis are specified in table 1.1

Notation	Definition
$\mathbb{N}$	Set of positive natural integers
$[\![n, m]\!]$	Set of integers between n and m
$\binom{n}{k}$	Binomial coefficient
$\|x\|$	Absolute value of x
$\|\mathcal{S}\|$	Cardinality of set $\mathcal{S}$
$\mathcal{S}\|\|\mathcal{S}'$	Concatenation of sets $\mathcal{S}$ and $\mathcal{S}'$
$\mathcal{S} \setminus \mathcal{S}'$	Substraction of set $\mathcal{S}'$ from set $\mathcal{S}$
$\log x$	Natural logarithm of x
$p(X)$	Probability of event X
$\mathbb{E}(X)$	Expected value of random variable X
$D_{KL}(p\|\|q)$	Kullback-Leibler divergence of distributions p and q
$\mathbf{A}$	Matrix notation
$\mathbf{v}$	Vector notation
$\mathbf{A}_{i,j}$	Value of matrix $\mathbf{A}$ at line i and column j
$\mathbf{A}^\top$	Transpose of matrix A

Table 1.1: Generic mathematical notations

Chapter 2

Thesis contributions

This chapter concludes this introductory part by providing a summary of contributions in section 2.1 and a list of publications in section 2.2.

2.1 Thesis summary and outline

This thesis studies acceleration of heavy workloads in smart device edge networks by providing observability and filtering of telemetry data, and a pipe-lining framework for processing throughput acceleration. It comprises 4 parts and 8 chapters, structured as follows.

Part I provides an introductory discussion, including a background on evolutions in application design and distributed computing, leading to the emergence of edge computing, are introduced, along with the constraints and limitations of this environment. Then, two interesting methodologies to improve the performance of the constrained smart device edge are presented, the cloud-out and edge-in approaches, which look at workload offloading and device improvement respectively to achieve better performance. Finally, a discussion is made about how increasing capacities in smart devices argues in favor of a hybrid approach, which leverages the multiplicity of edge compute by locally offloading computation.

Part II studies network state **observability** and **fault diagnosis** at the edge. Expert systems are computationally expensive to build and maintain, and lack scalability and inherent adaptability to unknown events or modifications in the topology of the edge network. In this context, chapter 3 (published in [52]) presents a data-driven method to extract intelligible selections of operational features from high dimensional network telemetry data, in order to provide visibility on the state of the edge, and facilitate fault diagnosis for operators. The presented method is based on extraction of information from meta-data information contained in feature names, through the example of Model Driven Telemetry (MDT) and the YANG modeling language. This is achieved by quantifying the information preserved in the **feature selection for fault diagnosis** process through a measure of cross-entropy, defined over a space of tokens contained in the YANG nomenclature. This measure leads to the definition of a selection quality score, in favour of selection preserving intelligible information for network operators.

With this semantic importance metric defined, chapter 4 (published in [53]) presents an evaluation of **semantic feature selection** in the context of fault diag-

nosis. This chapter advocates for semantic analysis in fault diagnosis from telemetry data, after having illustrated the shortcomings of three categories of data-driven methods. Purely data-driven mechanisms lack understanding of semantic importance within a feature set, and would benefit from additional domain knowledge. Using the methodology of chapter 3, this chapter explores the assumption that part of this additional knowledge can be extracted from meta-data. The proposed approach combines data-driven metrics and semantic information contained in the feature names to produce selections of features which best represent an underlying event. This study extends the cross-entropy based importance estimation, into an optimization method which joins semantic importance with data behavior. A benchmarking architecture is then introduced, to evaluate the benefits of this semantic analysis, and demonstrate the performance and robustness of semantic feature selection on different types of faults in network telemetry datasets, modeled with the YANG data modeling language. The results illustrate the interest of such a complementary meta-data analysis for data-driven fault diagnosis, and highlight the robustness of the studied approach against variations in the input feature set. The addition of a semantic analysis to a data-driven fault diagnosis process enables outstanding benefits in completeness of feature selections, as well as an important removal of parasite selection of features (*i.e.*, an increased precision).

Part III studies **heavy workload distribution** in edge environments, through the example of Deep Neural Network (DNN) inference throughput acceleration (published in [54]). DNN inference on streaming data requires significant computing resources to satisfy inference throughput requirements. However, latency and privacy sensitive deep learning applications, *e.g.*, in edge computing use-cases, cannot afford to offload computation to remote clouds because of the implied transmission cost and lack of trust in third-party cloud providers. However, within standard acceleration mechanisms, hardware acceleration can be onerous, and model optimization requires extensive design efforts while hindering accuracy. In chapter 5, DNN partitioning is presented as a third complementary approach, which consists of distributing the inference workload over several available edge devices, taking into account the edge network properties and the DNN structure, with the objective of maximizing the inference throughput (number of inferences per second). This chapter introduces a method to predict inference and transmission latencies for multi-threaded distributed DNN deployments and highlights issues linked in prediction accuracy linked to model run-time optimizations and heterogeneous hardware acceleration. With this representation of the inference behaviour, this chapter explicitly describes the potential service level objectives of DNN partitioning, *i.e.*, end-to-end latency, energy consumption, and inference throughput. After having identified this problem as a Mixed Integer Non Linear Programming (MINLP) problem, this chapter describes a branch and bound (B&B) optimization process with the objective of identifying partitionings which maximize the DNN inference throughput, while keeping computation on the edge. This method is compared with standard MINLP solvers to prove that B&B is better fit for edge computing contexts, because it solves simple cases quickly, and an early stopping mechanism is described to limit its complexity in corner cases.

In chapter 6, the DNN partitioning B&B solver is analyzed to quantify bounds on its performance and complexity. First, simulations are presented in a homogeneous

edge network environment (identical nodes and links) to explore the influence of the problem's input parameters on the achieved inference throughput and B&B complexity. The first simulations highlight the influence of the number of nodes and maximum number of split points on the performance and complexity, which leads to the identification of a bound on the necessary number of split points, hereby allowing network operators to limit the complexity of B&B. The second simulations depict the nodes and links on the network, showing that the effective number of iterations of B&B depends on a single parameter, *i.e.*, the link throughput to node processing rate ratio. Furthermore, these simulations identify three values of this parameter which delimit between regions with different behaviours in performance and complexity. This analysis has led to the definition of the *acceleration region*, which describes deterministic conditions on the DNN and network properties under which DNN partitioning is beneficial. Finally, experimental results are presented to confirm the simulations and show inference throughput improvements in sample edge deployments. The results also show of simple heterogeneous set-up in which the performance approaches the previously defined theoretical upper bound. The chapter ends in a summary of findings, which can be leveraged by operators and application developers to size their edge network and best take advantage of DNN partitioning, prior to the deployment.

In chapter 7, a **dynamic DNN partitioning** system is presented to study the robustness of DNN partitioning and its behaviour in unstable network environments. Edge networks being highly dynamic, this chapter illustrates the importance of timely re-evaluation of the optimal partitioning, since the previous chapters only presented a methodology for stable network conditions. First, this chapter illustrates the two principal benefits of dynamic re-computation: (i) the ability to compensate for the inference latency prediction inaccuracy by learning from run-time measurements, and (ii) the ability to adapt to different levels of persistent network changes. In this context,the presented simulation results show that DNN partitioning enables better robustness to perturbations, and dynamic partitioning allows the performance to remain optimal, even for important perturbations. Then, this chapter provides indications on a dynamic partitioning system design, and re-computation policies. Three policies are presented and compared: (i) a periodic re-computation policy, (ii) a reactive re-computation policy, which re-evaluates the partitioning when the inference throughput drops below a given threshold, and (iii) a proactive re-computation policy, which relies on the monitoring of telemetry data to trigger re-computations. This chapter concludes by identifying the link throughput to node processing rate ratio as a key factor in the choice of re-computation policy, because it defines the ratio between the B&B computational footprint, and that of actively monitoring the network telemetry.

Finally, part IV concludes this manuscript, and a summary in French is provided in appendix 8.

2.2 List of publications

The following contributions were published during the course of this PhD.

Journal publications

- Thomas Feltin, Léo Marché, Juan Antonio Cordero Fuertes, Frank Brockners, Thomas Heide Clausen, *DNN partitioning for inference throughput acceleration at the edge*, IEEE Access 2023 (chapter 5).

Conference or workshop publications

- Thomas Feltin, Parisa Foroughi, Wenqin Shao, Frank Brockners, Thomas Heide Clausen, *Semantic feature selection for network telemetry event description*, NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium (chapter 3).

- Thomas Feltin, Juan Antonio Cordero Fuertes, Frank Brockners, Thomas Heide Clausen, *Understanding Semantics in Feature Selection for Fault Diagnosis in Network Telemetry Data*, NOMS 2023 - 2023 IEEE/IFIP Network Operations and Management Symposium (chapter 4).

Part II

Understanding the state of the edge

Chapter 3

Information theory in network telemetry

Understanding the state of an edge device commonly involves a domain expert, who interprets a selected set of operational data to enable behavior monitoring, traffic engineering, anomaly detection, or fault analysis, based on domain knowledge and past experience. However, with the expansion of network sizes, the complexity of network structures, and the increasing volume of transiting data, monitoring the state of a network is a complex challenge.

Traditional measurement methods are separated into active and passive methods [55]. Active measurement methods consist of analyzing the behaviour of probe packets sent over the network, through mechanisms such as Ping [56] or Traceroute [57]. However, active measurement methods can create additional overhead, with an important impact on the observed network traffic. To prevent this "observer effect", passive methods consist of the exportation and analysis of telemetry data extracted to an external proxy, which aggregates data outside of the scope of the observed network. Passive mechanisms include NetFlow [58], sFlow [59], or IPFIX [60], which focus on the analysis of flows instead of individual packets, for better scalability. In order to unify the extraction of telemetry data from devices from different vendors, some efforts have focused on the standardization of the protocol and extracted data structure. Examples of such standards include the Simple Network Management Protocol (SNMP) [61] and SYSLOG [62], which define data structures to retrieve from heterogeneous devices. In 2006, the Internet Engineering Task Force (IETF) presented the Network Configuration Management protocol (NETCONF) [63], to improve the properties of SNMP, with additional capabilities such as structured representation of device configuration, using the Extended Markup Language (XML), different types of configurations, as well as tooling to facilitate concurrent access, validation of configurations, and event notification. In addition, the IETF presented the YANG data modelling language [64], which complements the NETCONF protocol, by representing the semantics of network configurations in a vendor-neutral way. The main advantage of NETCONF and YANG is the ability to define a network-wide configuration in a dedicated data-store, used to generate configurations for all devices, and removing the need for operators to push individual configurations on every device. Additionally, this enabled the development of Model-Driven Telemetry (MDT), a paradigm in which every device streams telemetry data to a dedicated server according to a network-wide configuration, instead of

the traditional intrusive pulling mechanisms. In sum, the emergence of MDT enables automated and frequent retrieval of all the available operational counters on a router, in a semantically consistent way through a collection of YANG modules [64].

The resulting telemetry datasets can be of large dimensionality, and distilling the information which best describes the device state can be challenging. Because of the dependencies between the different operational counters, the majority of these features is highly impacted in value when events occur in the network, *i.e.*, network or hardware-related faults which cause the global state of the device or network to change, *e.g.*, network loops, black holes, interface failures, memory leaks. Among all the thousands of features changing in value, only a few are really interesting for the diagnosis process. The majority of features are either (i) frequently changing in value independently of events, or (ii) only describing the consequences an actual event. For example, an interface failure will cause packet losses, route re-convergence, TCP connection changes, traffic changes, etc. which constitute the majority of changing features, while the features describing the actual root cause, *e.g.*, interface counts, will only be a few instances among the several thousands. An approach that takes all available features into account and distills those which are most descriptive of an event in an automated data-driven way is still missing.

3.1 Related work

Feature selection is the process of selecting the most important features in a dataset, in order to remove the irrelevant or redundant features [65]. Unsupervised feature selection performs this selection without the use of labels. Unsupervised feature selection methods can be categorized in three groups [65, 66]: wrapper, filter, and hybrid methods. Wrapper methods [67] select the subset of features which optimizes the result of a specific clustering algorithm, and have shown to be computationally expensive for high dimensionality problems [65]. Filter methods rely only on properties of the data to assign a relevance score and rank each feature in the dataset. The score of each feature can either be computed in isolation (univariate filter method), or in conjunction with the other features in the dataset (multivariate filter methods). The first filter method relying on information theory is sequential backward selection for unsupervised data [68], and is based on a measure of entropy with regard to the distances between features. SVD-Entropy [69] also uses the contribution of every feature to the entropy of the dataset to rank the features and find the subset with highest entropy. Similarly, [70] uses the concept of representation entropy to capture the amount of information contained in a selection. The idea behind [68–70] is that a high entropy translates to a balanced cluster structure, which implies that the features best represent the data. Multivariate methods such as Feature Selection using Feature Similarity (FSFS) [71] use statistical dependencies to further remove redundancy from a selection.

What the methods presented above have in common is their principal objective: finding the most relevant features, *i.e.*, features which contribute most to preserving the manifold structure of the original data, or features with highest or lowest correlation with the other features in the dataset [65]. The problem space of event description introduces a new constraint on the selected subsets, since they must not only preserve the information of the the original set, but also be descriptive of occurring events. In this problem space, the objectives in this study differ from those

of unsupervised feature selection, and are modified to include the two notions mentioned above. In that respect, methods from information retrieval in text provide tools to exploit the feature names, *e.g.*, TF-IDF [72], which estimates the importance of words in a document with respect to a corpus of documents, based on the compared occurrences of the words.

3.2 Statement of purpose

This chapter proposes a data-driven method to extract an intelligible selection of operational features, *i.e.*, features that can be easily understood and analyzed by a human, and which best describe an event occurring on an edge network, based on network telemetry data. This implies extracting feature importance from telemetry data, without the use of domain knowledge. Indeed, using domain knowledge would require a domain expert to annotate all features, which would not only imply annotating thousands of features for every possible event, but would also create a subjective view on feature importance. Every domain expert will look for different features in their diagnosis, which implies that features do not have an absolute importance value, *i.e.*, the ground truth cannot be defined in this problem space.

In order to design a data-driven analysis of the data, an assumption is made that, like in most telemetry datasets, features are labelled with a feature name which gives some information on the feature and/or the group they belong to. Raw feature names can be used as an indication of what functionality or what subgroup the feature refers to, *i.e.*, its semantic meaning. Therefore, this chapter uses feature names to help selection, in addition to raw data behavior.

The problem is naturally formulated as a feature selection problem [65], *i.e.*, the extraction of the most important features in a dataset, with revised objectives, to fit this problem space. While the literature focuses on preserving the overall information contained in the original dataset, this method generates selections which are (i) descriptive of an event, *i.e.*, contain features which see a significant change in value, and (ii) intelligible to a network engineer.

While the reaction of a counter to an event can be quantified from data behaviour, the intelligibility of a counter in a dataset, *i.e.*, how useful it is in helping a human explain the event, is strictly defined by domain knowledge. To this end, this chapter presents a metric to quantify the abstract notion of how intelligible a counter is in the dataset, based on the counter's rareness in the feature set. This metric is extended into a cross-entropy based metric to describe the overall intelligibility of a selection of features. The method then combines this estimation of intelligibility (domain knowledge) with a score for how strongly the features react to a change (data behaviour) to define an optimization score. This score allows the computation of *optimal* selections to help operators explain network events.

3.3 Chapter outline

The remainder of this chapter is organized as follows: Section 3.4 presents assumptions on telemetry data properties and preprocessing. Section 3.5 describes a method for estimating the relative importance of features, based on their feature

names. Finally, section 3.6 presents a metric to evaluate the semantic information contained in a selection of features, using a measure of cross-entropy.

3.4 Data description and preprocessing

This section presents the data particularities of telemetry datasets and preprocessing mechanisms in the context of fault diagnosis.

3.4.1 Telemetry data properties

Telemetry data takes the form of a time-series, with each feature representing the value of one particular sensor in a system over time. Specifically, in the context of network telemetry, the following data properties can be described.

High, variable dimensionality

In telemetry applications where the cost of an individual sensor, or of measuring an individual feature, is relatively low (as it is the case in networking, software engineering, IoT), telemetry datasets may be high-dimensional. In dynamic systems, dimensionality itself may change over time, as features appear or disappear. In particular, in network telemetry, performance of a network interface i may be described through n_i features or dimensions including interface i's byte count, data rate, queue occupation, etc. Enabling or disabling interface i on a device thus leads to an increase or decrease of the dimensionality of corresponding network telemetry dataset by n_i. Dealing with dataset *holes* that result from such variability may require additional data preprocessing.

Heterogeneity

The data values can be of different data types and formats, *e.g.*, in the network telemetry datasets used in this study, on an individual router, the numerical features can be positive incremental integer values ranging in the billions, *e.g.*, byte counts, or non-monotonic functions ranging from 0 to 1, *e.g.*, CPU consumption. Comparing data of different nature also requires additional preprocessing.

Aggregation level

Telemetry datasets are also heterogeneous in the aggregation level of each feature. Telemetry applications usually monitor complex systems, composed of different subsystems or services, such as network routers or mechanical systems composed of individual components and services. Telemetry data is usually composed of (i) features describing the state of a single element (referred as *individual* feature in this paper, *e.g.*, a single byte counter), and (ii) features which are aggregations of several sources of information (referred as *compound* features in this paper, *e.g.*, the total number of open connections on a router).

3.4.2 Telemetry data preprocessing

The properties presented in section 3.4.1 highlight the challenges of data-driven fault diagnosis. The high and varying dimensionality cause dimensions to appear and disappear from the feature set dynamically, without indication on their signification or relevance. Heterogeneity complicates the use of most data-driven approaches, which assume the identical nature of all features. Different aggregation levels imply different levels of importance in the feature set, which only stem from domain knowledge. However, with the assumption that the data around an event is fixed, it is possible to define a window around the time of the event where the data can be relieved of some of these properties. For example, differentiating the incremental features and performing min-max scaling can solve the heterogeneity problem. This differentiation can be performed in real-time by estimating which features are incremental during a bootstrapping period, the duration of this period being considered long enough to simply estimate that any monotonously increasing features is incremental. The dynamic dimension problem can be handled by padding missing values with zeros, which narrows down the dataset to a time-series of fixed dimension for further processing.

This preprocessing is not optimal: (i) min-max scaling does not handle unbalanced data well, (ii) the bootstrapping period can be too short and consider non-incremental data as incremental, and (iii) zero-padding dynamic dimensions can create artificial abrupt changes in the data which alter detection and diagnosis. Further optimization is outside of the scope of this paper since the presented preprocessing mechanism has proven to provide reasonable results in chapter 4.

3.5 Estimating semantic importance

This section presents an estimation of the relative importance of features for fault diagnosis, by exploiting available meta-data, which carries semantic information.

As described in section 3.1, the notion of relevance in the literature is most often linked to a measure of information contained in the resulting selection of features. In the context of fault diagnosis, this translates in a selection which needs to be as intelligible as possible, to help understand an event. The ideal selection in this problem space is one that distills the counters that are both describing the event, and meaningful to an operator, which needs to be defined.

Since this notion of meaningfulness is abstract and strictly depends on domain knowledge, this section proposes an approximation. This approximation allows the method to leverage the contextual information derived from the feature names, and offers a complementary analysis to what can be extracted from the data behavior.

3.5.1 Term Frequency-Inverse Document Frequency

Not all features in a telemetry dataset have the same relevance for the diagnosis of a given fault. Compound features are usually more relevant because they offer an aggregated perspective of the state of the system, compared to individual features which only describe a single component. For example, Bidirectional Forwarding Detection (BFD) [73] session counters (2 occurrences out of 6622 in the example

dataset[1]) are more meaningful to an operator than one of the many features counting the number of bytes sent counters on the router's interfaces (570 occurrences). More generally, it can be observed that semantic importance can be linked to a notion of rareness of a feature name, *e.g.*, with compound features containing the words `summary` or `total` in their feature name.

Standard methods to estimate semantic importance include the Term Frequency-Inverse Document Frequency (TF-IDF) [72] metric. TF-IDF quantifies the importance of a word (or term) in a document, within a corpus of documents; word *importance* is quantified as the relative frequency of the word in the document (term frequency), divided over an estimation of the information provided by the word in the whole corpus of documents (inverse document frequency), as follows:

$$\text{tfidf} = \text{tf} \cdot \log\left(N/\text{df}\right) \tag{3.1}$$

When applied to telemetry, documents translate into feature names, and words translate into the different *tokens* forming a feature name. Tokens can be individual words or groups of words that have collective semantic meaning in a feature name. For example, the feature name `up-interface-count` consists of three tokens (individual words): `up`, `interface`, and `count`.

In équation 3.1, tf is thus the number of times the token appears in the feature name, df the number of feature names containing the token, and N the dimension of the dataset. The importance of an entire feature name, *i.e.*, the semantic importance of one feature in the dataset, can be estimated as the average importance of its tokens.

Feature names are usually short and contain very few token repetitions. It can be estimated that the number of times a token appears in a feature name $tf \approx 1$, in which case the importance of a token can simply be approximated by the inverse frequency of its occurrence in the dataset. This observation shows that this metric is relying on the *discriminative* power of a feature name: the more unique a feature is in a dataset, the more information it contains when impacted by an event. Token importance is defined as a distribution over $\mathcal{T}$, the space of all existing tokens in the feature set. Considering $\tau \in \mathcal{T}$ a token, the importance of this token in the set is approximated as p:

$$p(\tau) = n(\tau)/T \tag{3.2}$$

with $n(\tau)$ the number of times the token τ appears in the set, and T the sum of the number of tokens in each feature name in the set.

3.5.2 Extension to MDT

The definition of frequencies in this section must describe the rareness of feature names in a dataset, *i.e.*, frequently occurring feature names must have a higher frequency value than less frequently occurring ones. Any method for generating frequencies with such properties is considered valid — this chapter will use the method described below.

As described in the previous section, a simple method for using descriptive feature names to quantify the rareness of a feature is to simply consider their occurrences in the dataset. In the case of MDT, the feature names are referred to as

[1] https://github.com/cisco-ie/telemetry

sensor paths [74], and are part of a 3-layered hierarchical name space. A sensor path corresponds to a branch in this topology, for example:

$$\underbrace{\texttt{tcp\_node\_statistics}}_{\text{token type 1}} : \underbrace{\texttt{interface\_1}}_{\text{token type 2}} : \underbrace{\texttt{bytes-sent}}_{\text{token type 3}}$$

In YANG, sensor paths can be parsed to extract three components: a module name, a key value array, and a leaf name, as described in [64]. Using the method presented in section 3.5.1, the rareness of a feature name is defined as the frequency of these individual components. These three *tokens* are instances within a *token type*. In the example above, sensor paths consist of three token types, and can be parsed into three tokens: `tcp_node_statistics` is a token within the token type of module names, `interface_1` is a token within the token type of key value arrays, and `bytes-sent` is a token within the token type of leaf names.

More generally, feature names can be parsed when their format is consistent, in order to make token types correspond to precise attributes. This distinction between module name, key value array, and leaf name, can be generalized to the distinction of K token types as the different attributes parsed in a feature name (in this case, $K = 3$). Within each type, the rareness of a feature name is defined as the frequencies of its tokens within their type in the set (giving K frequency values for a single feature name).

For a token type $0 < k \leq K$, T_k is the total number of unique tokens found among token type k in the entire set of feature names. For $0 < i \leq T_k$, $t_{k,i}$ the i-th individual token among the tokens of type k, and $n_{k,i}$ is the number of times token $t_{k,i}$ appears as the k-th token type in a feature name. In other words, this value is counting the occurrences of every unique token among the tokens of the same type. Finally, for $0 < k \leq K$ the frequencies $\{p_{k,i}\}_{0<i\leq T_k}$ are defined as $p_{k,i} = n_{k,i}/N$, where N is the total number of features in the set.

For example, the frequency associated with module name `tcp_node_statistics` (token type 1), the key value array `interface_1` (token type 2), or the leaf name `bytes-sent` (token type 3), will be the number of times each token appears divided by the total number of features in the dataset, giving one probability distribution p_k for each token type (3 in this case).

For each token type k, $p_{k,i}$ is a measure for how rare token $t_{k,i}$ is, within the token type k, and estimates how meaningful $t_{k,i}$ is to a network engineer (low values of $p_{k,i}$ translate to the token $t_{k,i}$ being most meaningful).

3.6 Semantic importance of a selection

With semantic importance of individual features defined, this section presents a method for defining the quality of a selection of features. For the remainder of this section, p refers to the distribution of tokens defined on the full set of feature names, and q refers to the distribution of tokens in the subset of features selected by a hypothetical feature selection method.

3.6.1 Quantifying selection quality

In the problem space described in this chapter, the objective is to produce selections which are both intelligible and semantically related to the event.

In the related work presented in section 3.1, a measure of entropy is often used, because it is correlated to balanced cluster structures in the data [65]. However, the objectives of the methods presented in the related work section are different, because they focus on reducing the size of the data to lower its impact, while an ideal selection in this study is one that is *specific* to a particular event, *i.e.*, corresponds to an unbalanced cluster structure. Entropy of a distribution p is defined as:

$$H(p) = -\sum_{\tau \in \mathcal{T}} p(\tau) \log p(\tau) \tag{3.3}$$

The more specific to a given functionality, or to a given element of hardware, the more information a selection will provide to an operator, and the lower the entropy value will be. On the contrary, if the selection is very diverse and contains features describing many different functionalities, interpretation will be more complicated , and the value of entropy will be high. In other words, if the entropy is low, the selection will be more *intelligible*, because it will be focused on a specific functionality.

3.6.2 Cross-entropy

Entropy does not capture the difference pointed out in section 3.5, *i.e.*, if the selected features are focused on a component which is rare in the original set, it will have the same score as if it was focused on an originally frequent functionality. The score for this feature selection method should be greater if the selection focuses on the rarest features in the dataset, estimated to be correlated to semantic importance.

In that respect, cross-entropy, *i.e.*, the relative entropy of a distribution compared to a reference, quantifies how focused/specific a selection is, along with how much it differs from the reference dataset. Cross-entropy captures how specific the information is in the selection, with the original distribution as reference distribution.

For the remainder of this chapter, it is assumed that the hypothetical selection process extracts a subset of features with a token distribution q from the original feature set with token distribution p. To describe an event, this subset is expected to preserve tokens from the original set which carry the most information, *i.e.*, rare token instances in distribution p. This is quantified through the measure of cross-entropy $H(p, q)$ between the initial and final token distributions p and q, defined as:

$$H(p, q) = -\sum_{\tau \in \mathcal{T}} p(\tau) \log q(\tau) \tag{3.4}$$

A high cross-entropy implies a low likelihood, and is related to semantic quality, *i.e.*, the ability for an operator to understand the selection. The advantage of using this metric is twofold: (i) it favors the selection of features composed of rare tokens, identified as carrying the most information, and (ii) it favors specificity in the distribution, which favors feature selections with a small number of tokens.

When applied to different token types, as described in YANG, for a given token type $0 < k \leq K$, cross-entropy is expressed as follows:

$$H(p_k, q_k) = -\sum_{i \in P} p_{k,i} \log q_{k,i} \tag{3.5}$$

If $p = q$, the cross-entropy value will simply be the entropy of the original distribution q. This means that random selections will have an average cross-entropy value of $H(q)$, while the scores of selections which focus on a specific functionality will be $H(p, q) > H(q)$. Additionally, the cross-entropy value will be greater if the focus is on a rare functionality, since the metric captures the difference in entropy between the two distributions.

Cross-entropy is very close to a divergence metric between two distributions. Compared to *e.g.*, the Kullback-Leibler (KL) divergence [75] D_{KL}, cross-entropy also indicates the specificity of the selection. When the difference between cross-entropy and the entropy of the original distribution is computed (to remove the constant component $H(q)$), it can be expressed as the sum of the KL-divergence and the difference in entropy between the two distributions, *i.e.*, the information gain $IG(q|\mathcal{S})$.

$$\begin{aligned} H(p, q) - H(q) &= D_{KL}(p||q) + H(p) - H(q) \\ &= D_{KL}(p||q) - IG(q|\mathcal{S}) \end{aligned} \tag{3.6}$$

Not only does this score describe the distance from the original distribution (divergence), it also provides an indication on how much information is preserved in the selection, compared with the original dataset (specificity).

Notably, the selection which maximizes the cross entropy value is just the one feature which contains the rarest tokens in the set. Having a single feature as selection is trivial to interpret, even more so when its tokens preserve the most information from the original set. However, in the context of fault diagnosis, this needs to be joined with feature contributions to the event, which is why cross-entropy is insufficient in itself, and needs to be complemented by a change amplitude metric, presented in chapter 4.

3.7 Summary of results

This chapter has proposed a general method for estimating feature importance, in the context of event description, in multivariate time-series data, through semantic information retrieved on the feature names only.

The proposed estimation of the intelligibility of a selection is based on TF-IDF, a measure of word importance in a corpus of texts, and is based on the occurrence of its components. In the context of telemetry, this metric is adapted to show the link between token rareness and semantic feature importance in telemetry datasets, illustrated through the example of MDT YANG datasets.

With this estimation of feature importance, this chapter has identified that a measure of cross-entropy on the token distributions in the features of a telemetry dataset represents the quality of a selection process. This selection quality score is an indicator of the information contained in a selection in the context of fault diagnosis, *i.e.*, how interpretable this selection is.

Chapter 4

Semantic feature selection

Fault diagnosis, *i.e.*, identifying the root cause of an event, has been studied in communication networks, manufacturing, maintenance of mechanical systems, transportation, and software engineering. By mimicking processes of human reasoning, expert or rule-based systems have proven to be useful for in-depth diagnosis [76]. Most efforts of expert systems for fault diagnosis rely on the definition of a state graph, representing the known and unknown states of the system, with defined transitions, depending on the available features [77]. Typical methods include probabilistic automata and Petri nets [78, 79]. However, such fault diagnosis systems present severe scalability and adaptability issues. They require an extensive modeling stage, with full knowledge of the fault behavior, which does not scale in large, relatively complex systems. In applications such as the IoT, where a variety of technologies and sensors interact with each other, or in network telemetry, where the dimension of the data changes with the network topology, telemetry data is often heterogeneous and of varying and high dimension, making it difficult to design a system which covers the entire fault behavior. Graph-based expert systems also imply high computational costs in the diagnosis process when the dimension increases [80]. Being hand-crafted and domain-dependent, expert systems also lack the ability to adapt to new, unseen data [76].

In this context, robust data-driven approaches allow to (i) avoid the cost of expert systems conception and maintenance, and (ii) leverage high dimensional telemetry data to robustly diagnose events with limited domain knowledge. Insight about the inner structure of the feature set (semantics, relations, relative importance) may overcome the absence or scarceness of explicit domain knowledge. Extracting and integrating that insight about inner structure and relationships within the feature set is thus a major challenge for improving the performance of fault diagnosis systems.

One way to provide a data-driven diagnosis is distilling a set of features that are of operational importance. Selecting original features can be a simple way to assist fault diagnosis while avoiding the modeling stage. Instead of presenting an expert with high dimensional data, feature selection narrows down the scope of investigation, based on a given metric, *e.g.*, individual amplitude of change, variance, or difference from a standard value.

4.1 Related work

In the literature, *explanation* refers to the identification of the input features, which contribute most to a model's decision [81,82]. Although this paper presents a method with the same objective, it does not consider any model, but rather isolated events at a given moment in a multivariate time-series. In this study, the process is entirely data-driven, and is rather considered as a features selection problem.

Typical feature selection methods [65] identify the most important features in a dataset without the objective of explanation or fault diagnosis. Feature selection is usually intended for dimension reduction. Classical dimension reduction methods such as Principal Component Analysis, Linear Discriminant Analysis, or t-Distributed Stochastic Neighbor Embedding [83–85] produce artificial dimensions, *i.e.*, dimensions which are combinations (linear or not) of the original features. For the purpose of interpretability in fault diagnosis, the returned features need to correspond to original dimensions.

Efforts in feature selection of original dimensions include wrapper methods [67], which select the original features with regards to a given clustering algorithm, and filter methods, which rely on data collections to define a relevance score, often based on metrics derived from entropy [68–70], or statistical dependencies [71] to capture the amount of information contained in the selected features. However, these selection methods have a different objective than those discussed in this chapter, which selects original features for explanation instead of computational efficiency.

Feature selection for fault diagnosis has been studied to detect faults in mechanical systems and modern process industries [86]. These applications consider small input dimensions compared to this study. Other selection mechanisms for diagnosis include explanation methods in Deep Learning applications. The *explanation* process aims at selecting the original features responsible for a classification decision. Computation of the Shapley values [81] or LIME [82] score the contributions of each input feature to a classifier, in order to better understand a decision process. Although the process of selecting original features for fault diagnosis is similar, the purpose of explainable AI methods is to describe the reason for a given classification (that is, for their own decisions), whereas the objective of this study is to describe the underlying data itself. A first specification and preliminary results of this method for fault diagnosis was presented in [52].

4.2 Statement of purpose

This chapter presents an evaluation of the semantic feature selection method, presented in chapter 3 [52] and summarized in figure 4.1, on network telemetry datasets. This method is a hybrid selection process which combines data-driven metrics and semantic analysis of meta-data. This approach produces a representation for network fault events, extracted from the telemetry available, that can be used for fault diagnosis. The contributions of this paper are the following:

- The demonstration that data-driven feature selection methods fail to identify semantic feature importance relationships.
- This introduction of a novel benchmark for evaluating the performance and robustness of selection methods for event diagnosis on telemetry data.

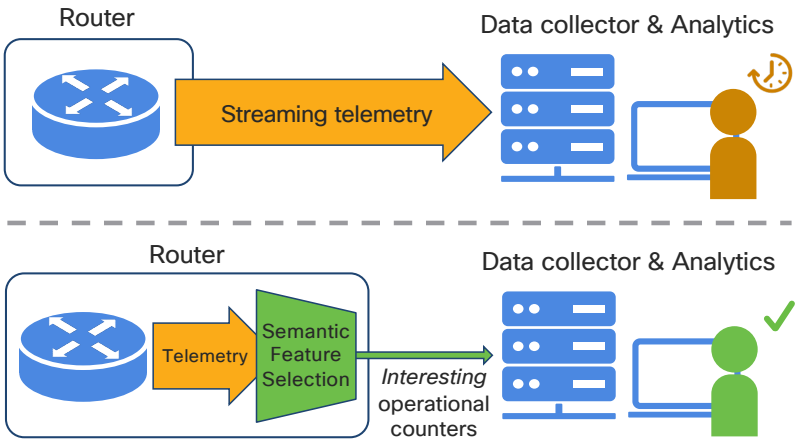


Figure 4.1: Semantic feature selection for fault diagnosis in network telemetry data. Instead of exporting large volumes of telemetry data to centralized servers, semantic feature selection can filter the interesting information for fault diagnosis.

- The presentation of benchmarking results, for both data-driven and semantic feature selection methods, on data retrieved from routers running the Cisco IOS-XR operating system, modeled with the YANG data modeling language [64]. This benchmark demonstrates both the performance and robustness of the semantic feature selection method for fault diagnosis.

4.3 Chapter outline

The remainder of this paper is organized as follows: section 4.4 presents the limitations of data-driven selection methods and highlights the need for additional information. Section 4.5 presents the semantic feature selection method based on the meta-data analysis presented in chapter 3. Finally, section 4.6 presents the experiment setup and results for performance and robustness evaluation on network telemetry datasets.

4.4 Data-driven methods

This section presents data-driven selection methods for fault diagnosis. These methods take an event time, t_0, as a given input[1], and return a ranking of the features with weights representing how much each feature changes within a window around the event time $[t_0 - w, t_0 + w]$. Throughout this section, the pre-processed time-series data is annotated as $\mathbf{S} = \{s_{n,t}\}_{n \leq N, t \leq T}$ with $s_{n,t}$ being the value of feature n at time t.

Three approaches are tested on network telemetry datasets to cover a range of data-driven change amplitude metrics: the univariate change amplitude method, which computes the change amplitude for every feature independently by looking at univariate time-series data (section 4.4.1), a linear multivariate change amplitude method, which considers N-dimensional data points for every time step (sec-

[1]Event detection itself and mechanisms for determining the time $t_0 = t(e)$ of an event e are outside the scope of this paper; a change-point detection method [87] could be used for this purpose.

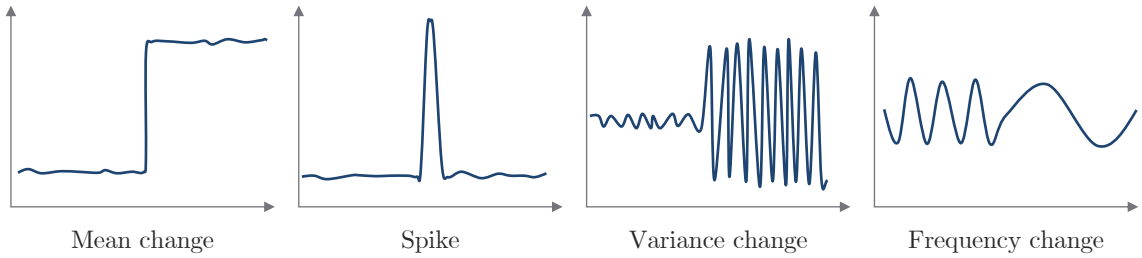


Figure 4.2: Change shapes present in telemetry datasets. For simplicity, the univariate change amplitude metric only considers mean and spike changes.

tion 4.4.2), and a non-linear N-dimensional method which identifies feature contributions to a non-linear classifier (section 4.4.3). A comparative analysis of the examined methods concludes the section (section 4.4.4).

4.4.1 Univariate Change Amplitude (UCA)

The univariate change amplitude is defined, for every feature, as the maximum value between the normalized mean value change between windows respectively before and after the time of the event, and the normalized spike amplitude at the time of the event. With $\mathbb{E}_{t\in[t_i,t_f]}(s_{n,t})$ being the temporal-mean value of the normalized data between times t_i and t_f, and A the value of the spike amplitude on a window around the time of the event $[t_0-\epsilon, t_0+\epsilon]$, for every feature n, the univariate change amplitude is expressed as:

$$\begin{aligned}\sigma(\mathbf{s}_n, t_0) = \max\Big(&\left|\mathbb{E}_{t\in[t_0-w,t_0]}(s_{n,t}) - \mathbb{E}_{t\in[t_0,t_0+w]}(s_{n,t})\right| \\ &\left|A - \frac{1}{2}(\mathbb{E}_{t\in[t_0-w,t_0-\epsilon]}(s_{n,t}) + \mathbb{E}_{t\in[t_0+\epsilon,t_0+w]}(s_{n,t})\right|\Big)\end{aligned} \tag{4.1}$$

The univariate change amplitude metric scores every feature independently by looking only at the univariate data in a given time frame. Given this data, when an event occurs in the system, the observed change can show several patterns, as shown in figure 4.2, *e.g.*, mean value change, variance change, spikes, frequency change. The metric needs to provide comparable values for different shapes or data type, *e.g.*, numerical, categorical, incremental or not. Limiting this metric to first moment statistics, *i.e.*, changes in mean value or spikes, simplifies the metric which can be used as a baseline for comparison with the more elaborate methods below.

4.4.2 Linear Discriminant Analysis (LDA)

The approach in this section considers N-dimensional data points instead of univariate time-series. The method used is the Linear Discriminant Analysis (LDA) [84], which computes the coordinates of the hyperplane which maximizes separability between N-dimensional data points before and after the event. With $\mathbf{s}_t$ defined as the N-dimensional vector of feature values at time t, the weights in the final ranking correspond to the coefficients of the hyperplane $\mathbf{v}^*$, defined as:

$$\mathbf{v}^* = (\boldsymbol{\Sigma}_- + \boldsymbol{\Sigma}_+)^{-1}(\boldsymbol{\mu}_- - \boldsymbol{\mu}_+)$$

$$\text{with} \qquad \boldsymbol{\mu}_{\pm} = \mathbb{E}_{t\in[t_0,t_0\pm w]}(\mathbf{s}_t)$$

$$\text{and} \quad \boldsymbol{\Sigma}_{\pm} = \sum_{t\in[t_0,t_0\pm w]} (\mathbf{s}_t - \boldsymbol{\mu}_{\pm})(\mathbf{s}_t - \boldsymbol{\mu}_{\pm})^{\top} \tag{4.2}$$

With this classification between points in windows around the event, the ranking will be less sensitive to outliers and represent general tendencies in the data. However, the classifier will be unable to discriminate between classes if the discriminatory, *i.e.*, this method makes assumptions on the distribution of the measurements being Gaussian within both classes.

4.4.3 Non-linear classifier

The chosen non-linear classifier is a random forest, trained to classify points before and after the event time, and the feature contributions are extracted by leveraging the SHAP methodology [81], which relies on the computation of the Shapley values.

Similarly to LDA, this ranking considers N-dimensional points at any given time t, without assuming a distribution over the two classes, which enables the consideration of non-linear relationships between points before and after the event time t_0. The SHAP methodology is originally designed for explainable AI, where methods are developed to find the original features which contribute most to a classifier decision. In order to apply this method, the binary random forest classifier is first trained to classify points before and after the event time, before applying the explanation methodology which ranks features by their contribution to the classifier predictions.

4.4.4 Limitations illustration

The three methods are applied on an example network telemetry dataset of 23650 individual features, describing the state of a router running the Cisco IOS-XR operating system, when an interface shuts down[2].

- *Univariate change amplitude*: Among the highest ranked features are mostly compound features such as interface counters, BFD sessions state counters, Border Gateway Protocol (BGP) neighbour counters, as well as individual features such as the last Routing Information Base (RIB) version, or negotiated intervals.

- *Linear discriminant analysis*: The highest contributing features are traffic counters, data rates and neighbour advertisement message counters.

- *Non-linear classifier*: The highest ranked features are exclusively packet counters and data rates related to the state of the interfaces neighbouring the shutdown interface.

[2]Because of the solutions verbosity, the 50 highest ranking features for each method can be found at https://github.com/tfeltin/sefset_results/blob/master/datadriven.md

Although all the selected features are either relevant to the interface shutdown, or linked to a consequence of this event, none of the methods above place the most important compound features first, *i.e.*, features representing the number of active interfaces. This suggest that the highest ranked features are highly changing in value, but do not necessarily contribute anything meaningful to the diagnosis.

Some features are more important than others in the diagnosis process, albeit identical from a data-driven point of view. For example, the feature counting active interfaces `up-interface-count` and the feature describing the last version of the RIB table `last-rib-version` have an identical behavior around the event, *i.e.*, a step from an integer value to another, yet `up-interface-count` can be considered as the most important indicator of an interface shutting down, while `last-rib-version` describes a consequence, as a part of the re-routing mechanism. From a purely data-driven approach, the two features are indistinguishable and are both included among the highest ranked features.

From this analysis, the conclusion is drawn that the described data-driven approaches cannot capture importance relationships between features, and additional information needs to be taken into account to identify important features in telemetry datasets.

4.5 Optimization problem definition

This section describes the semantic feature selection method, which jointly optimizes data-driven change and semantic information contained in the selected subset of features.

4.5.1 Defining the optimization objective

Considering a multivariate time-series of dimension $N > 0$, the frequencies describing token occurrences p and q are defined as in section 3.5. At a given time t_0, each uni-variate time-series $\mathbf{s}$ has an associated score $\sigma(\mathbf{s}, t_0)$ that quantifies the amount of change in the feature (cf. section 4.4). This section presents a method to find the subset of features $\mathcal{S}$ which best describes what is changing at a given time.

This can be expressed as an optimization process, which aims at selecting features which both maximize the data-driven change score, and the cross-entropy, through the product of both metrics (change score $\sigma(\mathbf{s}, t_0)$ and cross-entropy $H(p, q)$). The idea behind this optimization process is that optimal selections will both picture the change around a given time, with the change score, and diverge from the original feature set with high specificity, with the cross-entropy.

The product of both metrics is taken as optimization score $\mathcal{L}'$, and is defined as follows:

$$\mathcal{L}'(\mathbf{S}, p, q) = H\left(p, q\right) \frac{1}{|\mathbf{S}|} \sum_{\mathbf{s} \in \mathbf{S}} \sigma\left(\mathbf{s}, t_0\right) \tag{4.3}$$

where $\sigma(\mathbf{s}, t_0)$ is the univariate change amplitude score from section 4.4.1. When using several token types, the resulting scores can be aggregated by summing the values of token-specific cross-entropy, in order to take all token types into account, giving $\mathcal{L}'$, with $p = \{p_k\}_{0<k\leq K}$, and $q = \{q_k\}_{0<k\leq K}$ the distribution of tokens within a token type:

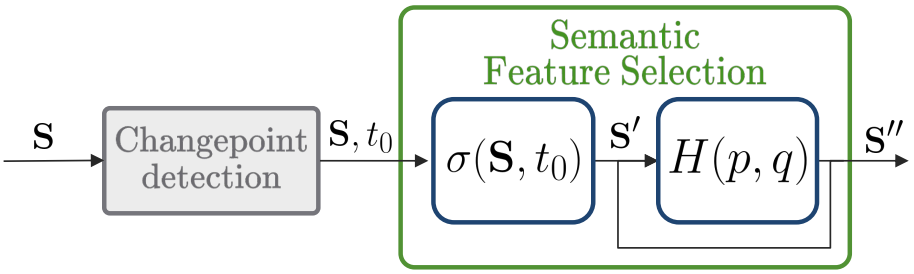


Figure 4.3: Flow chart of semantic feature selection for fault diagnosis. It is assumed that the diagnosis process is associated with a change-point detector (outside of the scope of this work), which produces time stamps t_0 of detected events from streaming telemetry data $\mathbf{S}$. The semantic feature selection uses a data-driven change amplitude metric $\sigma(\mathbf{S}, t_0)$ to define the optimization objective based on cross entropy $H(p, q)$ in équation 4.5.

$$\mathcal{L}'(\mathbf{S}, p_k, q_k) = \sum_{k=1}^{K} H\left(p_k, q_k\right) \frac{1}{|\mathbf{S}|} \sum_{\mathbf{s} \in \mathbf{S}} \sigma(\mathbf{s}, t_0) \tag{4.4}$$

However, looking for results which optimize this score results in fixed size selections. This can cause selections to be too long, *i.e.*, containing parasite information, or too narrow, *i.e.*, lacking some level of detail, for interpretability. In order to relax this constraint, and to have the method offer configurable amounts of features to describe the event, a regularization term is added to the score to penalize very small selections and arrive at the final definition of $\mathcal{L}$ [52]:

$$\mathcal{L}_\alpha(\mathbf{S}, t_0, p, q) = (1 - e^{-\frac{|\mathbf{S}|}{\alpha}}) H(p, q) \frac{1}{|\mathbf{S}|} \sum_{\mathbf{s} \in \mathbf{S}} \sigma(\mathbf{s}, t_0) \tag{4.5}$$

$$\mathcal{L}_\alpha(\mathbf{S}, p, q) = \left(1 - e^{-\frac{|\mathbf{S}|}{\alpha}}\right) \sum_{k=1}^{K} H\left(p_k, q_k\right) \frac{1}{|\mathbf{S}|} \sum_{\mathbf{s} \in \mathbf{S}} \sigma(\mathbf{s}, t_0) \tag{4.6}$$

where α is the regularization parameter, which aims at penalizing very small selections, and which can be used as a tuning parameter for the size of the selection. This regularization parameter defines how much smaller subsets are penalized. A higher value of α leads to selections with higher cardinality.

A flow chart of semantic feature selection showing how the optimization objective is defined to jointly maximize the data-driven change metric and the cross-entropy based semantic importance estimation is shown in figure 4.3.

4.5.2 Optimization process

The semantic feature selection algorithm is described in algorithm 1. The chosen optimization process is greedy: after initializing a selection by selecting the features with highest change score, at every, the impact of the addition/removal of each feature in the selected set is computed. Only those additions/removals that improve the optimization score ($\mathcal{L}$) are maintained/removed from the set. This process is repeated until no further additions/removals can improve the score.

Algorithm 1 Semantic feature selection

Input: $\mathbf{S}$ (time-series), t_0 (event time), α (regularization parameter), $I_{\max}$ (early stopping)

Output: $\mathbf{S}''$ (selection)

$\mathbf{S}' \leftarrow \sigma(\mathbf{S}, t_0)$ // Sorted by highest change amplitude
$I \leftarrow 0$ // Counting the number of iterations
$\mathbf{S}'' \leftarrow$ the first N_i elements of $\mathbf{S}'$
while $\mathbf{S}''$ keeps changing after an iteration, or $I < I_{\max}$ **do**
  $\mathbf{S}_{new} \leftarrow \{\}$
  for $\mathbf{s}$ in $\mathbf{S}'$ **do**
    if $\mathbf{s} \in \mathbf{S}' \setminus \mathbf{S}''$ and $\mathcal{L}_\alpha(\mathbf{S}'' \parallel \{\mathbf{s}\}, p, q) > \mathcal{L}_\alpha(\mathbf{S}'', p, q)$ **then**
      $\mathbf{S}_{new} \leftarrow \mathbf{S}_{new} \parallel \{\mathbf{s}\}$
    else if $\mathbf{s} \in \mathbf{S}''$ and $\mathcal{L}_\alpha(\mathbf{S}'' \setminus \{\mathbf{s}\}, p, q) < \mathcal{L}_\alpha(\mathbf{S}'', p, q)$ **then**
      $\mathbf{S}_{new} \leftarrow \mathbf{S}_{new} \parallel \{\mathbf{s}\}$
    end if
  end for
  $\mathbf{S}'' \leftarrow \mathbf{S}_{new}$
  $I \leftarrow I + 1$
end while

The input variables are the multivariate time-series data $\mathbf{S}$, the event time t_0, the regularization parameter α, and the maximum number of iterations for early stopping $I_{\max}$. Firstly, the change score $\sigma(\mathbf{S}, t_0)$ is computed for the multi-variate time-series $\mathbf{S}$, as depicted in figure 4.3, and the features are ranked in descending order. The initializing step selects the N_i features with highest change scores as the initial selection, with N_i defined depending on the use-case ($N_i = 500$ in the network telemetry application used in section 4.6). Then, at every step, the selection mechanism performs the two previously mentioned operations: (i) for every feature that is not in the selection, the score is computed for the selection with this extra feature included, and every feature which improves the score of the current selection when added is appended to the current selection; then, (ii) the same operation is performed, by instead trying to remove each feature in the selection, and every removed feature which improves the score is removed from the current selection. The optimization process stops when no addition or removal improves the score. If this criterion is not reached after $I_{\max}$ iterations, the process stops and returns the current selection with the notification that an optimum was not reached.

At the end of the optimization process, the selected features are ordered by their contribution to the score (computed by leave-one-out), in order to have the most important features first.

4.5.3 Illustration

The output of the selection method is a list of original input features which maximizes the score defined in equation 4.5.

To illustrate the results, the process was run on MDT datasets where the feature names follow the nomenclature of an associated YANG model [64]. The data was extracted from a router through MDT collections in a lab environment, where typical network events were inserted, such as interface failures, routing loops, traffic black

holes, etc.

The results showcased below were obtained by running algorithm 1 on a dataset created, by retrieving several MDT collections with a cadence of 10s over several periods, in a lab environment. The resulting set consists of over 40 thousand features describing the router state, with triggered enables/disables on interface 10 to trigger interface shutdowns, enables/disables of a bidirectional forwarding detection (BFD) session to trigger failure of BFD sessions, and addition of routing entries to trigger routing loops. The time-stamps at which the events occur are known, and the change score (σ) for each uni-variate time series was computed as the univariate change amplitude metric described in section 4.4.1.

The output of semantic feature selection on a routing loop, an interface shutting down, and breaking a BFD session, for $\alpha = 1$, are the following [3]:

Routing loop

- `Cisco-IOS-XR-ipv4-io-oper:ipv4-network/nodes/node/statistics/traffic [node-name=0/1/CPU0]output`
- `Cisco-IOS-XR-ipv4-io-oper:ipv4-network/nodes/node/statistics/traffic [node-name=0/1/CPU0]hopcount-sent`
- `Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/ latest/protocols/protocol[interface-name=HundredGigE0/1/0/34 protocol -name=IPV4_UNICAST]bytes-sent`
- `Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/ latest/interfaces-mib-counters[interface-name=HundredGigE0/1/0/34] bytes-sent`
- `Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/ latest/generic-counters[interface-name=HundredGigE0/1/0/34]bytes-sent`
- `Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/interface-xr/interface [interface-name=HundredGigE0/1/0/34]bytes-sent`

The first two features in this list are the ICMP output and hop count exceeded features, followed by traffic counters on the interface which is seeing the incoming traffic from the loop.

Admin Interface 10 shutdown

- `bfd_summary::session-state_down-count`
- `bfd_summary::session-state_up-count`
- `interface-summary::admin-down-interface-count`
- `interface-summary::up-interface-count`

[3]The sensor paths in the rest of the paper were abbreviated for the sake of readability, but the detailed features names can be found in the publicly available dataset https://github.com/cisco-ie/telemetry/tree/master/11.

- bfd_counters:HundredGigE0/0/0/10:hello-transmit-count
- bfd_session:HundredGigE0/0/0/10:negotiated-local-transmit-interval
- bfd_session:HundredGigE0/0/0/10:negotiated-remote-transmit-interval

The selected features are the ones counting acitve interfaces, and the BFD session monitoring the impacted interface.

Breaking BFD session

- bfd_summary::session-state_up-count
- bfd_summary::session-state_down-count
- bfd_counters:HundredGigE0/0/0/16:hello-receive-count
- bfd_counters:HundredGigE0/0/0/16:hello-transmit-count
- bfd_session:HundredGigE0/0/0/16:negotiated-remote-transmit-interval
- bfd_session:HundredGigE0/0/0/16:negotiated-local-transmit-interval

The selected features are all BFD session counters.

4.5.4 Discussion

The resulting selections for $\alpha = 1$ are both intelligible and descriptive of the corresponding events. The selection process has allowed the selection of less than 10 of the most important features, out of the initial tens of thousands, and these features can be categorized as meaningful to an operator (ICMP hop counts, BFD sessions counts, and active interface counts). With the removal of the many counters that can obscure a user's analysis, it will be easier for an operator with this condensed view to infer an explanation of the event.

Figure 4.5.4 shows the size of the selections compared to the original number of features for the different types of events contained in the dataset. Figure 4.5.4 also displays the number of changed features around the time of the event, and the number of features whose values have changed by more than 95% in absolute value.

As a comparison, the simple method of extracting the top 10 features with highest value of σ_i (normalized difference in mean value on windows of 10 points before and after the timestamp) for the interface 10 shutdown would return:

Interface 10 shutdown

- fib-statistics:0/0/CPU0:incomplete-adjacency-packets
- bgp:ipv4:performance-statistics_vrf_update-generation-prefixes-count
- bgp_default-vrf_afs-process-info:ipv6:global_label-version
- bgp_default-vrf_afs-process-info:ipv6:global_last-rib-version

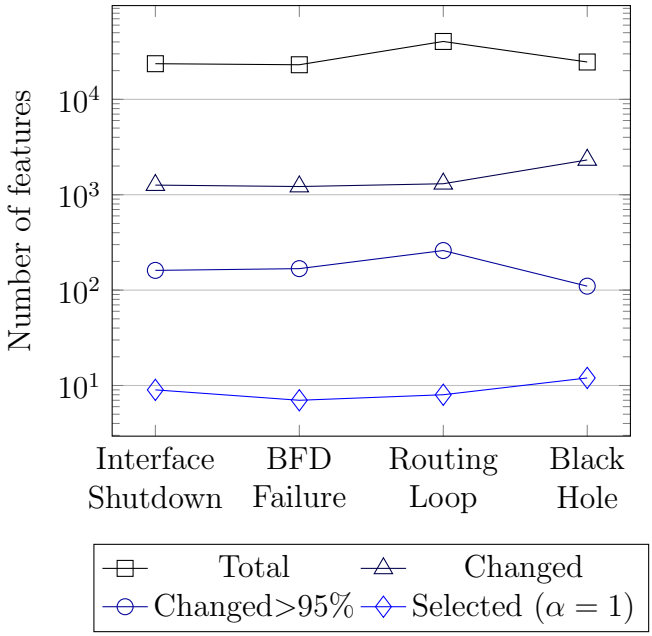


Figure 4.4: Number of selected features with $\alpha = 1$ compared to the original number of features and the number of changing features (logarithmic scale) around given events in the dataset. "Changed" describes the number of features which change in value around the event, and "Changed>95%" are the ones which see a change of more than 95% of their absolute value.

- `bgp_default-vrf_afs-process-info:ipv6:local-paths-freed-num`
- `bgp_default-vrf_afs-process-info:ipv6:local-paths-malloced-num`
- `bgp_default-vrf_afs-process-info:ipv6:paths-freed-num`
- `bgp_default-vrf_afs-process-info:ipv6:paths-malloced-num`
- `bgp_default-vrf_afs-process-info:ipv6:rib-acked-table-version`
- `bgp_default-vrf_afs-process-info:ipv6:rib-bgp-version`

Although these features are actually caused by the event (re-routing caused by the interface shutting down), they are unrelated to the nature of the event itself. This specific dataset contains 8% of Virtual Routing and Forwarding (VRF) related sensor paths. As discussed in section 3.5, because of their high occurrence in the dataset, they are estimated to convey less meaning to the network engineers. As an indication, in the selection made by this method with $\alpha = 1$, the change scores (σ_i) of the selected features ranged from 51th to 109th highest change score. This implies that the contribution of cross-entropy to the optimization score resulted in the selection of the features which carry most information (*e.g.*, interface counts) and the removal of those that did not, among the features changing most in value. This observation confirms the contribution of both components of the optimization score ($\mathcal{L}$): while the change score (σ_i) allows the method to select the counters that see a change in value around the event, the cross-entropy ($H(p, q)$) selects those which are most meaningful to a network engineer (*i.e.*, rare in the dataset).

The α parameter further allows the tuning of the verbosity of the selection. For example, with lower values of α, the selection for the interface 10 shutdown becomes:

Admin interface 10 shutdown: ($\alpha = 0.47$)

- `bfd_summary::session-state_down-count`
- `bfd_summary::session-state_up-count`
- `interface-summary::admin-down-interface-count`
- `interface-summary::up-interface-count`

Admin interface 10 shutdown: ($\alpha = 0.1$)

- `bfd_summary::session-state_down-count`
- `bfd_summary::session-state_up-count`

A higher value of α can allow for a more detailed view (which can hint at the details of the events, *e.g.*, locality) whereas lower values can result in scarce selections.

4.6 Benchmark

Comparing this feature selection for fault diagnosis method to existing solutions is difficult because the literature lacks a dedicated evaluation. This section therefore proposes a benchmark to address this issue, and identify the added benefit of a semantic analysis of meta-data for fault diagnosis. This approach also aims to evaluate the robustness of the method with variations in the collected input features, in order to quantify the dependency of the semantic method on the input feature set.

4.6.1 Datasets

The datasets used in this benchmark are extracted from several devices in a Clos-topology lab environment with simulated traffic and inserted events[4]. Each dataset contains 20 to 40 thousand individual features depending on the device. The datasets in the benchmark each contain one single inserted event. The time of the event is known to the selection mechanism. The purpose is to evaluate every selection output in an isolated way.

Four injected events are used in this benchmark and for each event, data was collected from different devices : (i) interface admin shutdown, on the two devices at the ends of the disconnected link, (ii) BFD failure (filtering BFD message to trigger a failure), (iii) black hole (removing FIB entries to cause silent packet drops), collected on the concerned device, and (iv) a routing loop (by adding static routes), with data collected from the three devices concerned by the loop.

[4]Details on the topology in https://github.com/cisco-ie/telemetry

4.6.2 Metrics for feature selection

Objectively evaluating a feature selection for a diagnosis process is complicated: the feature importance during diagnosis is subjective and operator dependent. With this observation, defining a metric which quantifies precisely how far the output is from an ideal is impossible. Although the precise ground truth is intractable, the metrics defined in this section aim to get an assessment of how the method is performing.

The idea is to define an upper and lower bound on what the acceptable outputs are for this method, based on the described event. This paper defines the following two metrics that will act as such:

- *Precision*: defined as the proportion of features in the output selection that are related to the input event. Among all features, those related to an event are defined as a pre-established subset which are deemed relevant to select when an even occurs. This metric quantifies the relevancy of the selected features with regards to the input event.

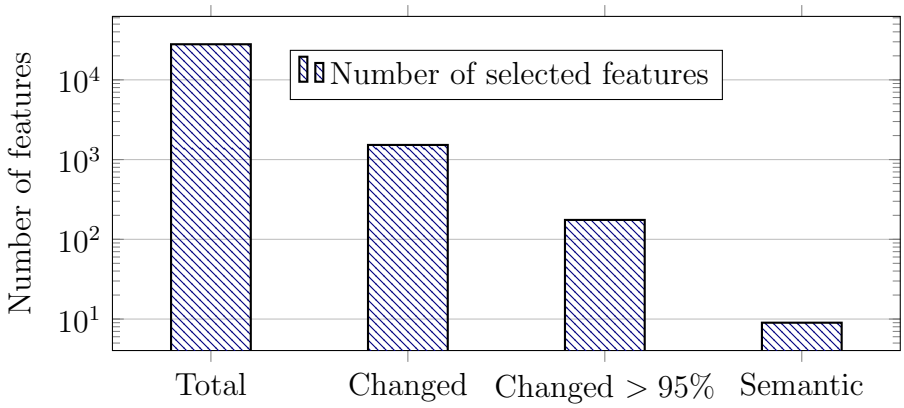
- *Completeness*: Within the list of all acceptable features to describe an input event, a subset of these features are essential to diagnosis. Completeness quantifies whether the output contains this very minimal requirement, *i.e.*, completeness is the proportion of this minimal subset which is present in the output.

Along with performance, this paper defines one other metric to evaluate the robustness of the method. As the selection method highly depends on the input data, over-fitting to either the raw data or the feature set is a possibility. Therefore, the following metric is used:

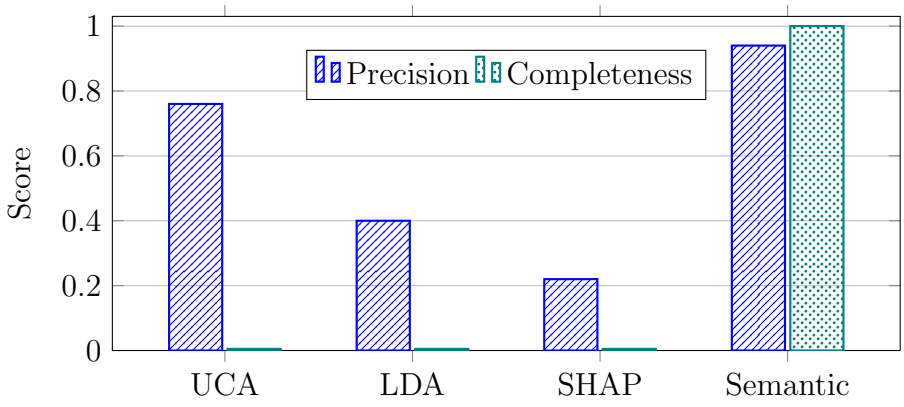
- *Robustness*: quantifies the degree of variation in the output when the input collection (feature set) is modified. Intuitively, since the importance estimation highly depends on the probability distributions defined in section 3.6, it is expected that variations in the feature set may cause significant variations in the selection. The robustness evaluation aims at identifying precisely which type of variation in the feature set is associated with which degree of variation in the final selection.

4.6.3 Ground-truth definition for evaluation

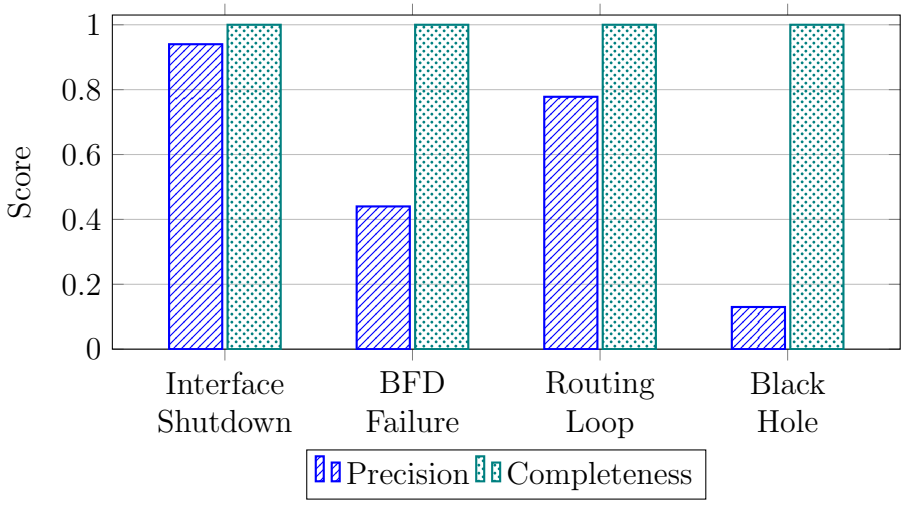
For each event type, computing the performance and robustness metrics depends on the definition of a set of features which act as ground truth for both precision and completeness. The precision ground truth feature set should contain all features which are linked to the event, and the completeness ground truth feature set should contain the minimal set of features expected from the selection. For simplicity in this network telemetry use-case, the two feature sets are defined as a collection of regular expressions. This benchmark defines a ground truth for four event types in the context of network telemetry: interface shutdowns, routing loops, BFD session failures, and black holes. Figure 4.6 shows the ground truth definition for datasets using YANG models.



(a) Comparison of number of reduced dimensions



(b) Precision and Completeness scores of methods for an interface shutdown



(c) Precision and Completeness scores

Figure 4.5: Comparative benchmarking results between semantic and data-driven feature selection methods for event diagnosis in telemetry data. Figure 4.5(a) shows the amount of dimensions reduced. Figure 4.5(b) shows the benchmarking scores of the data-driven methods described in section 4.4 and compares them to semantic feature selection on an interface shutdown. Figure 4.5(c) shows the benchmarking results of semantic feature selection on the four events in the benchmark.

Event type	Regular expressions
Interface Shutdown	**P :** `interface.*summary / HundredGigE0/0/0/N` **C :** `interface-count`
BFD Failure	**P :** `bfd` **C :** `bfd.*summary.*count`
Routing Loop	**P :** `icmp / bgp / hopcount / bytes-(sent|received) / ipv4-io-oper.*traffic` **C :** `hopcount`
Black Hole	**P :** `rate / load / icmp / ipv4-io-oper.*traffic / connections-(established|accepted|closed)` **C :** `unreachable-received`

Figure 4.6: Ground truth definition for the four event types contained in the benchmark, for network telemetry data modeled with YANG. Regular expressions are defined for precision (P) and completeness (C).

4.6.4 Robustness evaluation

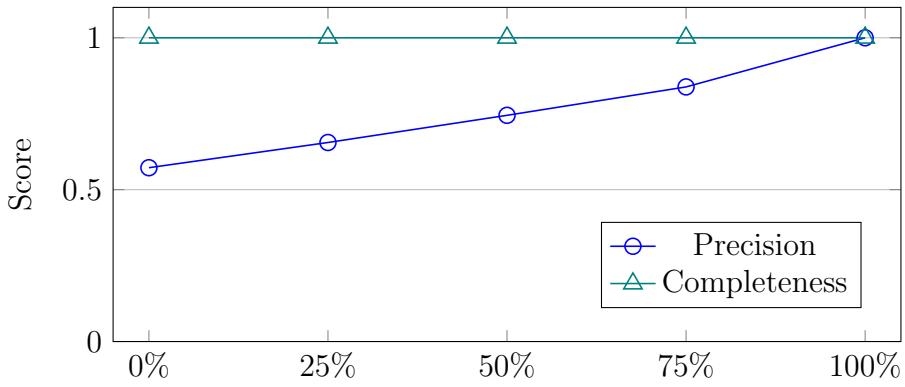
The space of all possible input subscriptions is too large to be fully enumerated. The robustness evaluations presented in this study target two scenarios: (i) incomplete feature sets, *e.g.*, caused by collection errors, and (ii) variations in token distributions, *e.g.*, caused by different subscriptions or operators.

Random removal of entire modules

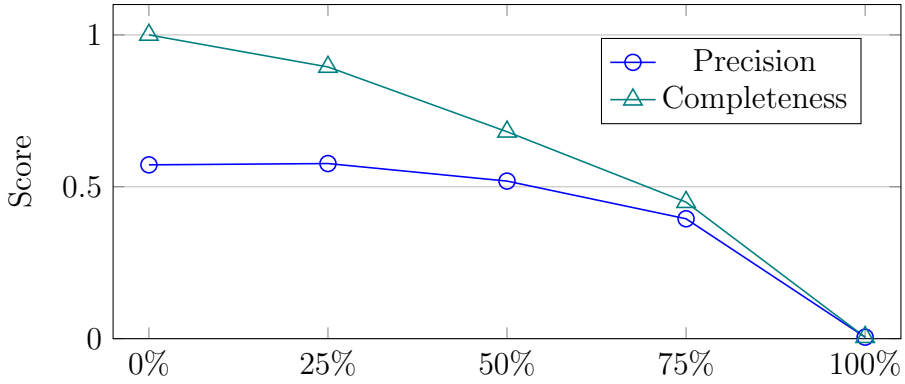
From the list of subscriptions contained in the datasets, this method measures the impact of removing all the data contained in one or more modules on the selection output. Different quantities of removal are tested, *i.e.*, 25%, 50%, and 75% of modules. Two metrics are extracted from this evaluation based on whether the removed features belong to the ground truth defined in section 4.6.3 or not: sensitivity, *i.e.*, score variations when ground truth related features are removed, and consistency, *i.e.*, score variations when non ground truth features are removed.

Changing feature name distribution

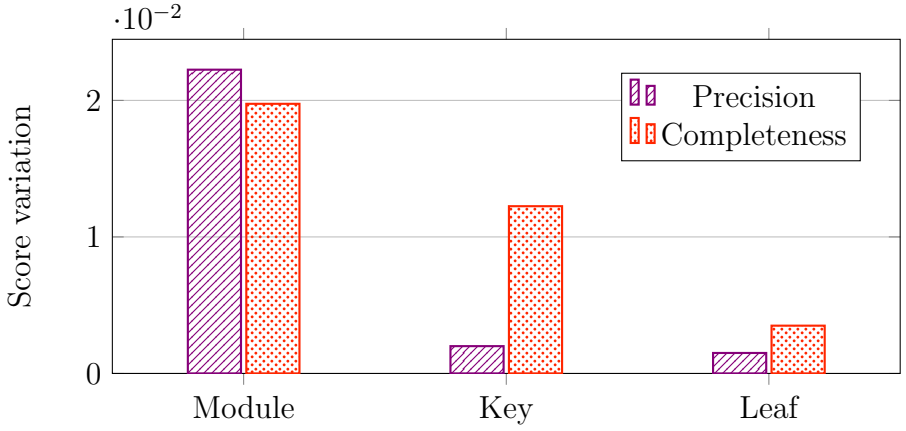
This methodology willingly makes frequent tokens rarer, as frequency is the determining factor in the importance estimation. Since YANG paths can be split into three types of tokens, *i.e.*, modules, keys, and leaves [64], this process is done independently for each token. The top 5%, 10%, 25%, 50% most frequent tokens in each type are made rare, by keeping only one feature among all the ones containing the given token. Robustness scores are computed for precision and completeness for each token type, for a more granular view. The robustness scores are the average score variations caused by altering one token frequency.



(a) Consistency to removal of modules not contained in the ground truth



(b) Sensitivity to removal of modules contained in the ground truth



(c) Error induced by altered token distributions.

Figure 4.7: Robustness evaluation. Figure 4.7(a) (resp. figure 4.7(b)) shows the evolution of the precision and completeness of selections when removing modules which are not included in the ground truth (resp. which are included in the ground truth). Figure 4.7(c) shows the impact of altering token distributions for three token types in YANG, as described in section 4.6.4.

4.6.5 Results and discussion

Figure 4.5 shows the number of selected features and their performance evaluation results on the benchmark for the four previous event types, and compares them with data-driven methods. The results vary depending on the event, with perfect completeness scores and varying precision[5]. Good completeness with lower values in precision, such as is the case for the black hole evaluated in the benchmark, corresponds to verbose selections with parasite information, but still the right essential indicators of the event. Overall, the semantic analysis identifies the 10 to 20 counters which best represent an event in datasets of 20 to 40 thousand individual counters. Additionally, the comparison with a data-driven method shows the effect of semantic analysis. Although the data-driven methods find features which are linked to the event, they are unable to find the most important feature which are related to an event, as hinted in section 3.4, and confirmed by the completeness score.

Figure 4.7 shows the results of the robustness evaluation. Figure 4.7(a) and 4.7(b) show the robustness evaluation scores from the first method, *i.e.*, the achieved variations in precision and completeness when removing features from the input set. As disclosed by the first robustness test, when removing 25% to 50% of the input features, the selections are still relevant. Additionally, with around 60 modules in the studied feature sets, when dividing these results by the number of removed modules, it can be argued that removing a single module has very little impact overall.

Figure 4.7(c) shows the robustness evaluation using the second method, *i.e.*, willingly altering the distribution of tokens in the feature set. The main takeaway is that the impact of drastically changing a token's frequency is almost insignificant on the final selection, which is unexpected. Changing the frequency should mean changing its importance estimation, and break the logic of the semantic analysis, yet the selections stay similar. One potential cause is the intrinsic logic that exists in the combinations between the different tokens in the naming model. When removing all the features that contain one given leaf name, the modules and key values consequently removed from the dataset are not randomly selected. This robustness evaluation shows that making the most frequent tokens rare does not impact the importance estimation severely enough to invalidate the semantic analysis.

4.7 Summary of results

Telemetry data often being heterogeneous, high dimensional, and of varying dimensionality, traditional expert systems lack adaptability and require extensive design and maintenance efforts. Data-driven approaches can be studied as lightweight and robust solutions. In that regard, semantic feature selection for diagnosis is a general solution offering a hint as to which original features best represent the event.

This chapter has illustrated how purely data-driven feature selection methods for fault diagnosis can be inefficient, and unable to identify the most important features to describe an event. Although such importance relationships are defined by domain knowledge, this chapter studies an approach for estimating these semantic

[5]The complete selection results can be found at https://github.com/tfeltin/sefset_results/blob/master/results.md

importance relationships by studying the meta-data information contained in available feature names. This semantic analysis produces more complete and precise selections, while significantly reducing the number of features to analyze.

With the elaboration of a benchmark for evaluating feature selection methods for fault diagnosis, this study has also shown the performance and robustness of a semantic approach on network telemetry datasets, with feature names derived from associated YANG models. The benchmarking results show the added improvement of semantic analysis compared to data-driven methods, with an average precision score 1.5x higher than data-driven methods and a significant completeness score improvement (0% for purely data-driven methods). Additionally, this study has evaluated the method robustness to variations in the monitored dataset, *i.e.*, removed and altered modules. This evaluation has illustrated robustness of the semantic analysis against strong variations in the input feature set, indicating its ability to capture semantic relationship between features independently of the input feature set. This study has shown experimental results on four event types, for telemetry datasets using the YANG modeling language. A potential extension of this work could include testing with a higher variety of fault types, and potentially other modeling structures (*e.g.*, SMI, OpenConfig), to confirm its applicability.

The method produces intelligible selections, which can ease the interpretation of events by a network operator, while allowing the exploitation of all the available counters in the dataset. This prevents operators from having to tediously hand pick which features to monitor, in cases where the data is of high dimension. The proposed method is equipped with a single value regularization parameter, allowing the adjustment of the desired output verbosity. This allows to find the selection size which best fits a given use case, by distilling the right amount of information. This regularization has shown to generate selections of less than ten highly meaningful features out of thousands in the dataset, for events with different root causes.

Although a univariate change amplitude and simple token distributions may produce intelligible results for network telemetry datasets, they need to be further evaluated, in cases where (i) the changes in the data are more complex, or (ii) the relative rareness of a feature isn't the best indicator of its relevance. These assumptions were taken based on the general observation that it is usually the case in network telemetry datasets. These two metrics are nevertheless defined independently from the general method to compute optimal selections, and they can be adapted to any other problem space.

Part III

Heavy workload distribution

Chapter 5

DNN partitioning at the edge

Connected devices generated an estimated 2.5 quintillion bytes of data every day in 2020[1]. In this context, Artificial Intelligence (AI) is one of the technologies that can best cope with the always growing amount of produced data, in a range of fields such as object detection in computer vision, facial recognition, speech recognition, natural language processing, or autonomous driving. AI, and specifically Deep Learning (DL), requires, in addition to large amounts of data, significant capabilities in processing power and memory, which makes cloud computing the *de facto* hosting solution for AI workloads. Consequently, in 2019, 96% of AI tasks were run in the cloud[2].

Some AI applications have strong latency constraints, *e.g.*, in autonomous driving, manufacturing monitoring, or any real-time inference involving a real world interaction. For example, average response times in autonomous driving are required to be under 100 milliseconds [1], making round trip times to the cloud too long for such applications. Similarly, some applications may introduce a significant link usage on the network, *e.g.*, in the case of high quality video processing in computer vision, which prevents them from running in the cloud. Data privacy policies or data protection regulations may also prohibit data from leaving specific environments. This encourages AI deployments in edge computing, which consists of relocating computation tasks from data centers closer to edge devices, *i.e.*, in proximity to the data sources.

In comparison with public clouds, the edge is a resource-constrained environment with important limitations [88], and developing AI for the edge is therefore intrinsically different from developing AI for the cloud. Analyzing data close to its source can be challenging because of lower device capacities, inelasticity of the available resources, and heterogeneity of edge networks. Lowering the computing capacity implies lowering the achievable inference throughput, further complicating relocation of heavy workloads from the cloud to the edge. In some applications, the inference throughput, *i.e.*, the achievable inferences per second of a DNN, can be linked to the DNN accuracy, *e.g.*, with object tracking in video streams, where dropping the inference rate from 15 to 5 inferences per seconds has shown to lower the F1 score[3] of an object detector by 10% [89]. In other words, deployments with

[1] According to the *Data Never Sleeps* annual study from Domo https://www.domo.com/learn/infographic/data-never-sleeps-8

[2] According to a survey from Nucleus Research Inc., 96% of Deep Learning based applications ran in the cloud in 2019. https://d1.awsstatic.com/whitepapers/Deep%20learning%20on%20AWS.pdf

[3] The F1 score is a measure of a model's accuracy on classification tasks.

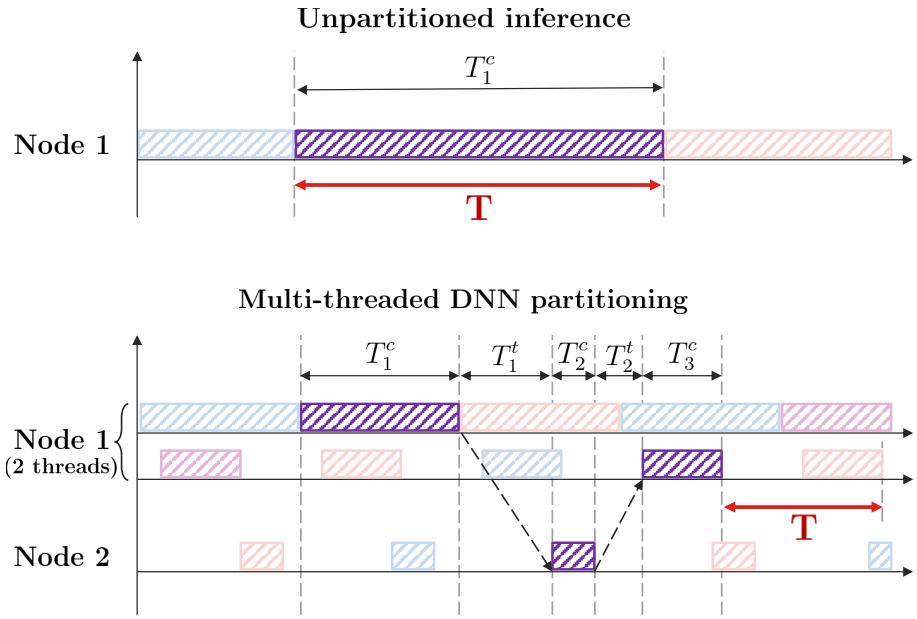


Figure 5.1: Timeline of multi-threaded inference partitioning over 2 devices compared with unpartitioned inference. The figure shows the computation (T_i^c) and transmission (T_i^t) latencies. The inference throughput is improved by partitioning, and bound by the slowest element in the network (**T**).

low inference throughput can cause critical information loss, *e.g.*, loss of detections in video surveillance applications.

There are two main methods to meet performance requirements on constrained edge networks: model compression or hardware acceleration. *Model compression* consists of reducing and optimizing neural networks to a light-weight, often underperforming, version of the model. Standard model compression methods include pruning [90], *i.e.*, removing unnecessary parameters in the model, quantization [91], *i.e.*, reducing the allocated memory to store the model, and knowledge distillation [92], *i.e.*, training a smaller neural network with knowledge extracted from a larger one. Model optimization requires extensive design efforts and can be an impediment to model accuracy. *Hardware acceleration* on edge devices implies identifying and adding hardware which can be both power efficient and light-weight, while still being able to run inference with sufficient performance. There are several types of candidate hardware for acceleration at the edge, varying in efficiency and specificity, but standard processing units fail to meet expectations and dedicated hardware such as ASICs have been found to be unpractical and expensive [93].

DNN partitioning is a complementary method to these two, for accelerating inference by leveraging the multiplicity of existing devices on edge networks to distribute the inference computation – which can be used alone, or in conjunction with the two other methods. DNN partitioning consists of considering a neural network as a pipeline to segment into partitions, and distributing these partitions on edge devices. The placement of these partitions is based on both the DNN and the underlying network characteristics. DNN partitioning relies on the identification of split points, which are points in the model graph where the model is separated into partitions. During run-time, partitions are run sequentially, each sending intermediary inference results to the next partition. This allows each partition to start comput-

ing the next input data while the other devices continue processing the offloaded one, hereby improving the inference throughput, as shown in figure 5.1. For the remainder of this chapter, DNN partitioning is illustrated through the example of real-time inference on video streams.

5.1 Related work

Methods for DNN partitioning, derived from mobile edge-cloud offloading, seek to optimize varying performance indicators, such as computing latency, energy consumption, resource utilization, cost, or throughput. DNN partitioning relies on the association of one or several of these metrics to define an optimization goal, and a method which exploits this metric to find partitioning schemes. For example, Neurosurgeon [94] seeks a single split point, keeping the first partition at the edge and offloading the second partition to the cloud, to minimize latency and energy consumption. Applications in IoT have also considered the joint partitioning and offloading of several DNNs to optimize energy, delay, and/or cost, using different solvers such as SPSO-GA [95] combining Particle Swarm Optimization (PSO) and Genetics Algorithm (GA), or DDPQN [96] which uses Deep Reinforcement Learning to find partitioning schemes. Other examples include DINA [97], which defines an Integer Non Linear Programming (INLP) problem, and uses a matching theory based solver, to optimize delay and resource utilization in fog networks. Methods optimizing inference throughput include DNN surgery [98], which uses the lower size of intermediary DNN layer outputs to partition the inference computation between the edge and the cloud. Both algorithms create two partitions, one at the edge and the other in the cloud. Other studies in the field of mobile edge-cloud [99] also include multi-threaded computation in the cloud to further accelerate the overall inference throughput. Relying on the cloud for the second partition inference computation assumes good network connectivity, which can be uncertain with the poor connectivity of some isolated edge deployments. Edgent [100] adapts the previous methodology to mobile devices and available edge servers, relaxing the constraint of offloading to the cloud. This method is linked to a specific training and model architecture, and still limits the partitioning to two partitions. Other methods have considered multiple split points, *e.g.*, IONN [101], which describes the problem of partitioning to minimize latency and energy consumption, also taking into account the time to upload the models to edge servers.

These methods are either interacting with remote clouds, or are limited in the nature of the partitioning, *e.g.*, required to split in exactly two parts, with or without multi-threading. None of the listed DNN partitioning studies considers both multiple split points and multi-threading.

5.2 Statement of purpose

This chapter presents a distributed inference framework to maximize the inference throughput of real-time DNN computation on streaming data, with multiple split points and multi-threaded partitioning. The contributions of this chapter are the following:

- A model for computing and transmission latencies of a distributed DNN, through which the expected inference throughput of a given partitioning scheme can be estimated.
- Formulation of the optimization problem for DNN partitioning, implementation of a branch and bound solver for this problem, and evaluation of its complexity.

For the remainder of this chapter, to avoid confusion, the partitioned deep neural network will be referenced as *DNN*, and the term *network* will denote the underlying physical communications network.

5.3 Chapter outline

The remainder of this chapter is organized as follows. Section 5.4 presents background of Deep Learning and DNN representation in this study. Section 5.5 details the inference and transmission latency prediction methods, required for section 5.6, which defines the optimization problem, as well as the branch and bound algorithm to take partitioning decisions.

5.4 Background

This section presents background on artificial intelligence at the smart device edge, as well as deep learning and feed-forward deep neural networks.

5.4.1 AI at the smart device edge

Since the adoption of cloud computing, one of the technology trends that has driven developments in edge and cloud is Artificial Intelligence (AI). Artificial Intelligence encompasses all tasks where computers exhibit intelligence and has been tightly linked to advances in the location of data processing. This technology has shown tremendous capabilities, *e.g.*, the ability to detect objects in images for video surveillance or self-driving cars, process speech and text for translation and virtual assistants, or process audio streams. In some applications, those capabilities even exceed humans, *e.g.*, composing entire symphonies, defeating humans in strategic games such as chess or go, or writing eloquent speeches and texts.

Artificial Intelligence has shown tremendous capabilities in a large range of fields such as object detection in computer vision, facial recognition, speech recognition, natural language processing, autonomous-driving, or home electronics. However, AI and Deep Learning algorithms require large computing capabilities, making the cloud the obvious hosting solution for such workloads. Nevertheless, the drawbacks of cloud computing, detailed in Section 1.1.3, apply very well to AI. Latency can be paramount in applications such as autonomous driving, operations on factory floors, or any real-time inference involving a real world interaction. For example, average response times in autonomous driving are required to be below 100 milliseconds [1], making round trip times to the cloud too high for such applications. Similarly, the link usage over the WAN can be significant, *e.g.*, in the case of high quality video processing for computer vision, making it impossible to reasonably rely on the cloud.

Privacy can also be essential in fields such as surveillance or cybersecurity. Although AI has gained in popularity in the cloud, the edge is the logical place for running inference with any of the previous requirements.

Although better fitted to serve latency and privacy sensitive AI applications than the cloud, the edge still remains a resource constrained environment with important limitations [88]. The three main constraints at the smart device edge are the following:

- *Power efficiency and form factor*: Edge devices supporting AI workloads need to be both powerful enough to comply with large computational requirements, and efficient enough to comply with the limited energy and form factor requirements of the edge. For this purpose, several computation units can serve this purpose. Although general purpose processing units, *i.e.*, CPU and GPU, are more broadly used for simplicity, they can be inefficient for long lasting workloads, mainly in energy consumption. Specific solutions, *e.g.*, DSP-based processors, are more power efficient but are also less flexible during development.

- *Computational optimization*: Limitations in computing resources along with requirements in latency imply strong optimizations in the development of algorithms for the edge.

- *Privacy*: Security and privacy are essential to the deployment of edge applications. Edge devices are by nature exposed to threats and closer to sensitive data. Implementing security in restrained edge devices is a challenge.

Developing AI for the edge is intrinsically different than developing AI for the cloud, where efficiency, optimization, and privacy are not an issue. Heavy Deep Learning workloads designed to process sensitive data are even more difficult to develop with power efficiency, computational optimality, and privacy in mind.

5.4.2 DNN inference

The term Artificial Intelligence (AI) is a broad denomination that references intelligence exhibited by machines [102]. It refers to any process that learns from its environment and produces conclusions or decisions to maximize its probability of success at some goal. Capabilities of AI include detecting, classifying, and segmenting objects in images and videos, understanding and translating human speech, competing in strategic games, piloting self-driving vehicles, and interpreting complex data [103]. AI has shown abilities that out-perform humans, *e.g.*, , with GPT-3 deemed able to write better than most humans [104], or AlphaZero developed by DeepMind which has shown superhuman capabilities at chess and go[4].

Within the field of Artificial Intelligence, Machine Learning (ML) focuses on processes with the ability to learn without being explicitly programmed to [105]. Machine Learning algorithms rely on statistical analysis of data to automatically improve through experience. Machine Learning generally relies on a performance

[4] https://www.deepmind.com/blog/alphago-zero-starting-from-scratch

metric and consists in searching through the space of candidate programs to find the one which best optimizes this score [106].

Machine Learning algorithms can be divided into four main categories: Supervised Learning, Unsupervised Learning, Semi-supervised Learning and Reinforcement Learning [107].

- *Supervised Learning*: Given a set of labeled input data and their corresponding output, supervised learning is the task of learning a function that maps an input to an output based on example input-output pairs [102]. Typical functions of supervised learning include regressions which fit data to a function, or classifications which separate and categorize data.

- *Unsupervised Learning*: Given a set of unlabeled data, the unsupervised learning category covers several exploratory tasks to analyze the data, *e.g.*, by identifying trends, groups, or anomalies. Typical functions of unsupervised learning include dimension reduction, visualization, clustering, feature learning, or anomaly detection.

- *Semi-supervised Learning*: This category covers methods in between the two previous categories, *i.e.*, with some labeled data and some unlabeled data. The high-level objective of semi-supervised learning is to improve the performance with unlabeled data compared to what the performance would be, had it been trained only on the labeled data. Typical functions of semi-supervised learning include text classification, fraud detection, or labeling data [107].

- *Reinforcement Learning*: Methods which fall into this category learn what actions to take, *i.e.*, map situations to actions, so as to maximize a numerical reward signal [108]. A learner is not explicitly told what to do, but rather learns through a process of trial an error which actions will maximize a final reward. The main applications of reinforcement learning include training agents in robotics, autonomous driving, or manufacturing.

Depending on the application, algorithms can use one or several of the methods in this list to best fit specific use-cases.

Within the field of Machine Learning, Deep Learning (DL) refers to methods relying on the use of Deep Neural Networks (DNNs), *i.e.*, artificial neural networks or related machine learning methods containing more than one hidden layer. Most of Deep Learning solutions involve the use of an artificial neural network, which were originally inspired by neurons in the visual perception regions of the brain. One of the earlier references to mention a deep learning architecture was in 1967 [109] and described a supervised feed-forward multi layer perceptron. In the 1990s, the back-propagation technique and convolutional networks were allied to the neural network architecture to identify hand-written digits [110] (at the time, the training took three days). The term itself was introduced later in 2000 [111]. It was not until the 2010s that DL gained traction, first in speech recognition, then in object detection in images, for the ability to capture complex relationships in high dimensional in data. Deep Learning is estimated to represent the majority of AI applications in 2022, with more than 75% of organizations using DNNs for applications that could use classical methods[5].

[5] https://gartner.com/smarterwithgartner/gartner-predicts-the-future-of-ai-\technologies

One way to look at deep learning methods is to consider "*techniques for machine learning in which hypotheses take the form of complex algebraic circuits with tunable connection strengths*" [102]. These algebraic circuits are usually organized into layers, which form steps in the computation. The term deep refers to algorithms consisting of more than one layer. An example computation graph structure is shown in figure 5.2.

There are two main DNN structures: feed-forward neural networks and recurrent neural networks. Both types of DNNs can be represented as computation graphs, with the main difference being that feed-forward networks can be represented as direct acyclic graphs (DAGs), while recurrent neural networks may contain cycles. Recurrent neural networks are built for sequences of data, where each data point has a potential dependency with previous data points in the sequence, while feed-forward neural networks do not consider interactions between points in a data sequence.

The computation graph of a feed-forward neural network can be represented as a DAG, with each node in the graph representing a layer, *i.e.*, a linear operation on its inputs followed by a non linear operation, called an activation function. For layer L_i in the DAG, x_i the inputs for layer L_i, $\mathbf{\Phi}$ the activation function, and $\mathbf{w}$ the weights and biases associated with layer L_i, the output of the layer L_i is expressed as:

$$\mathbf{x}_{i+1} = \mathbf{\Phi}\left(\mathbf{w}^\top \mathbf{x}_i\right)$$

It is worth noting that the activation function $\mathbf{\Phi}$ being non-linear is what allows DNNs to model complex relationships between the individual dimensions of the input. If the activation functions were linear, the model would be equivalent to a concatenation of linear operations, and represent a linear function.

With a layer defined as a unit in the computation graph of feed-froward networks, developing and training a DNN consists in defining a DAG, and adjusting the weights of these operations to fit the input data to a desired output inference. This adjustment is achieved by defining a loss function, which is often some distance between the expected and actual output, and use back-propagation to disperse this error to all weights in the hidden layers of the network.

Convolutional layers differ from the fully-connected layers presented above. Fully-connected layers on high-dimensional data, *e.g.*, images, has a considerate impact on computation and memory consumption. Instead of connecting every dimension in two adjacent layers, convolutional layers use smaller weights matrices as sliding windows over the input. This method allows to both save on computation resources, and leverage the spatial patterns in the input data, because kernels will focus on smaller portions of the input. Other notable layer types include pooling, which consists in extracting a single value from a subset of values in the previous layer, *e.g.*, max-pooling or average-pooling, taking respectively the maximum or average value from a subset of its input values.

The remainder of this paper will consider the case of feed-forward DNNs[6].

5.4.3 Hardware acceleration and model optimization

This section describes optimization mechanisms that enable AI and mainly Deep Learning at the edge. Given the computational and memory footprint of DL work-

[6]Feed-forward DNNs are well suited for inference on video streaming data.

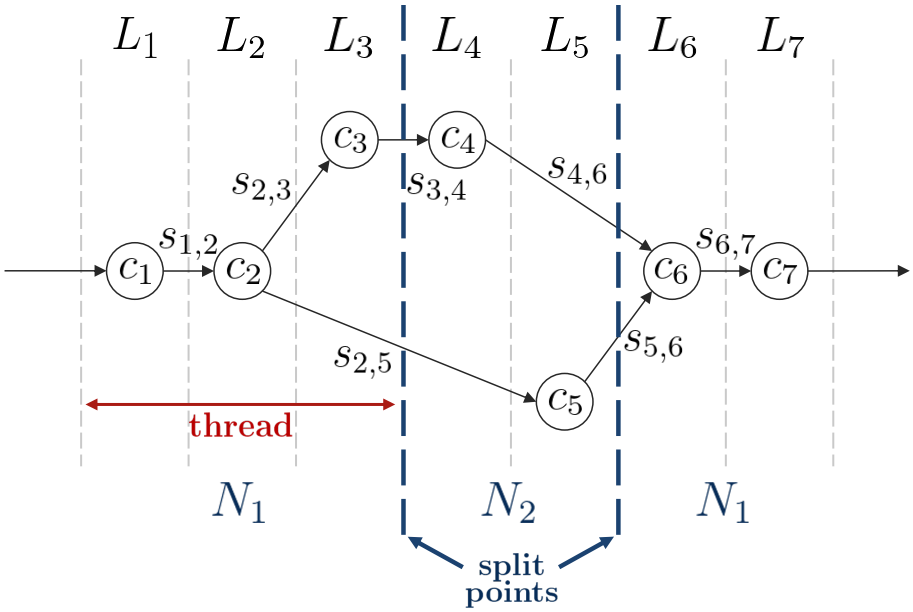


Figure 5.2: Example computation graph of a feed-forward neural network. Each layer L_i runs c_i operations, sends $s_{i,j}$ amount of data to the following layer, and is placed on a node N_p (see section 5.5.1).

loads, deploying such algorithms at the edge requires to go through optimization steps to fit on constrained edge devices. There are two main directions taken by developers to meet performance requirements on the edge, (i) hardware acceleration and (ii) model optimization.

Public clouds are equipped with large numbers of specialized hardware to serve DL tasks. Accelerating DL workloads on edge devices implies finding the similar hardware which can both be power efficient and lightweight, while still able to run inference with good enough performance. There are several types of candidate hardware for acceleration at the edge, varying in efficiency and specificity [93]. Graphical Processing Units (GPUs) are the more flexible solution, while power intensive, *e.g.*, the NVIDIA Ampere architecture[7]. Field Programmable Gate Arrays (FPGAs) are re-programmable integrated circuits, customizable through a specific programming language, which are more efficient but less flexible than to GPUs [112]. Application Specific Integrated Circuits (ASICs) are integrated circuits built for a specific application, which are most efficient in computation and power consumption but can only serve a given task, *e.g.*, Google's Tensor Processing Unit (TPU) [113], or the Arm Ethos set of Neural Processing Units (NPUs) [114]. Overall the simplest way to add capacity to an edge device would be to leverage CPUs, GPUs or DSPs, *e.g.*, most ML applications on mobile phones run on CPU. These standard processing units pain to meet expectations, but dedicated hardware such as ASICs prove to be unpractical and expensive.A performance benchmark of different chipsets and acceleration hardware is evaluated in [115] on mobile devices.

While hardware acceleration looks at upscaling the edge device, there are model optimizations that can optimize the model to fit on edge devices. These optimizations fall into several categories: (i) pruning, (ii) quantization, and (iii) knowledge distillation. Pruning methods [90] focus on removing unnecessary parameters in the

[7] https://www.nvidia.com/en-us/data-center/ampere-architecture/

DNN with limited impact on the model performance. These methods include a pruning phase which identifies and removes unnecessary parameters and connections in the DNN, and a fine-tuning phase, which recovers from the performance loss caused by the pruning. To further reduce the memory footprint of DNNs, quantization approximates the weights of the model by lowering the allocated bit width, instead of using floating-point numbers [91]. For example, using these methods, Deep Compression [116] is able to compresses the AlexNet model to a memory footprint 35 times smaller without any loss of accuracy. Knowledge distillation [117] mimics teacher-student learning relationships and consists in training a small low-precision neural network with knowledge extracted from a larger model. Knowledge distillation systems are composed of a formulation of knowledge, a distillation algorithm, and a teacher-student model pair. Knowledge distillation can significantly improve the performance of different DNNs, and be performed online or offline, with one or more teacher-student models. Other mechanisms include self-knowledge distillation [118–120] and mutual knowledge distillation [121]. The resulting model is a version of the faster student model which outperforms the larger teacher model [117].

Hardware acceleration and model compression have shown to provide important performance improvements, and DNN partitioning offers a complementary solution to further increase the inference throughput at the edge.

5.5 Distributed inference modeling

This section presents a model to represent DNN and network properties, in order to predict computation latencies, transmission latencies, and the final inference throughput of a given partitioning solution.

5.5.1 DNN and network representation

A feed-forward DNN of N layers is modeled as a direct acyclic graph (DAG) $\mathcal{G}_A = (\mathcal{L}, \mathcal{E})$ with $\mathcal{L} = (L_1, ..., L_L)$ being the layers of the DNN. The edges $(L_i, L_j) \in \mathcal{E}$ are the connections between layers L_i and L_j. Each layer L_i has an associated compute consumption c_i, measured in the number of floating-point operations required to compute a forward pass through the layer. Edges (L_i, L_j) are assigned a weight $s_{i,j}$ corresponding to the size of the data transiting between layers L_i and L_j in bytes.

The physical network is modeled as a fully connected graph $\mathcal{G} = (\mathcal{N}, \mathcal{V})$ where $\mathcal{N} = (N_1, ..., N_N)$ is the set of compute nodes and $\mathcal{V}$ is the set of links between nodes. It is assumed that compute nodes $N_1, ..., N_N$ have processing rates $\eta_1, ..., \eta_N$, respectively, measured in floating-point operations per second. Finally, the link throughput between two adjacent nodes N_a and N_b is denoted as $\theta_{a,b}$, measured in bytes per second. Every node N_i is connected to itself with infinite throughput to represent the loopback link, and links are assumed to be symmetrical, *i.e.*, $\theta_{a,b} = \theta_{b,a}$.

Partitionings are defined as maps $P : \mathcal{L} \to N$, *i.e.*, a partitioning assigns a node number to each layer in the DNN. A partitioning can be described as a matrix $\mathbf{P}$ of dimension $(N \times L)$, N being the number of nodes on the network and L the number of DNN layers and , with $\mathbf{P}_{a,i} = 1$ if layer L_i is placed on node N_a, 0 otherwise. With the example given in figure 5.2, $L = 7$ layers and $N = 2$ compute nodes, the displayed partitioning with layers $\{L_1, L_2, L_3, L_6, L_7\}$ on node N_1, and layers $\{L_4, L_5\}$ on node N_2, is represented as:

Symbol	Description
$\mathcal{G}$	The communications network graph
$\mathcal{G}_A$	The feed-forward network graph
$\mathcal{L}$	The set of layers in the DNN graph
$\mathcal{E}$	The set of connections between layers in the DNN graph
$\mathcal{N}$	The set of compute nodes
$\mathcal{V}$	The set of links between compute nodes
L	The number of layers in the DNN
N	The number of compute nodes in the network
L_i	The i-th DNN layer
N_a	The a-th compute node
c_i	The compute consumption of layer L_i in FLOPs
$s_{i,j}$	The size of the data transiting between layers L_i and L_j
η_a	The processing rate of node N_a
$\theta_{a,b}$	The link bandwidth between nodes N_a and N_b
$\mathbf{P}$	The partitioning matrix
$\mathbf{H}$	The inverse processing rate matrix
$\mathbf{c}$	The layer consumption vector
$\mathbf{S}$	The transmission size matrix
$\mathbf{\Theta}$	The inverse link bandwidth matrix
$T^c(N_a)$	The inference latency on node N_a
$\mathbf{T}^c$	The inference latency matrix
$T^t(N_a, N_b)$	The transmission latency on link (N_a, N_b)
$\mathbf{T}^t$	The transmission latency matrix
C	The inference throughput
p_s	The partial placement list of size s
$\tilde{p}_s$	The padded version of placement list p_s
S	The maximum allowed number of split points
α	The processing rate to link throughput ratio

Table 5.1: Summary of mathematical notations

$$\mathbf{P} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix} \tag{5.1}$$

Given a partitioning, a *thread* is defined as a group of consecutive layers between two split points, run sequentially on the same node. In the example above, node N_1 will run two threads, the first containing layers $\{L_1, L_2, L_3\}$ and the second one containing $\{L_6, L_7\}$.

With these notations defined, the remainder of this section derives a closed expression of the inference throughput achieved by a given partitioning, as a function of the inference (section 5.5.2) and transmission (section 5.5.3) latencies.

5.5.2 Inference latency prediction

Given a partitioning, this section looks at inference latency prediction on an isolated node. The latency induced by the computation of a layer L_i with consumption c_i, to be computed on node N_a, with processing rate η_a is expressed as $T_i^c(N_a) = c_i/\eta_a$. Given a set of layers $\mathcal{L}' \subset \mathcal{L}$ across all threads running on node N_a with processing rate η_a, the inference latency can be expressed as:

$$T^c(N_a) = \eta_a^{-1} \sum_{L_i \in \mathcal{L}'} c_i \tag{5.2}$$

This expression is a first order approximation of an inference latency estimation based on the number of floating-point operations (FLOPs) required to process the DNN, *i.e.*, the number of multiply-add operations in the model.

Estimation of inference latency on edge devices is complicated by run-time optimizations. Existing solutions such as nn-Meter [122] predict inference latency based on FLOPs. Other solutions rely on a look-up table with pre-computed inference times for latency inference. For example, BRP-NAS [123] uses a pre-trained graph convolutional network to predict inference latencies, while taking run-time model optimizations into account.

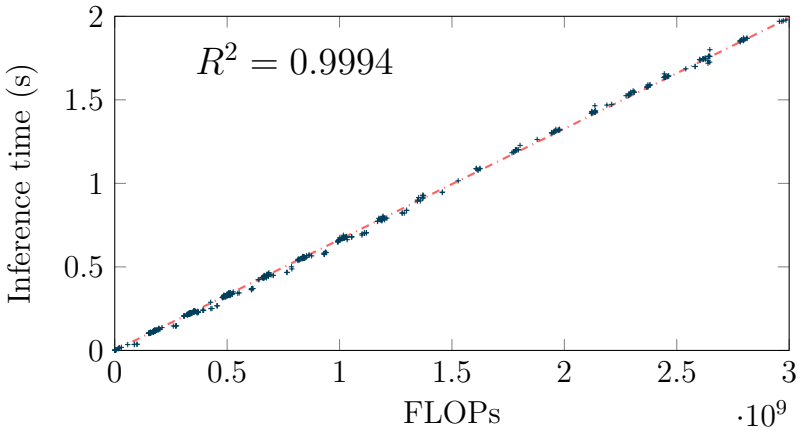
Inference latency prediction is further complicated by its difference in behavior depending on the underlying hardware. As an example, figure 5.3 depicts the dependency between FLOPs and inference latency of a YOLOv2[8] [124] model on CPU and GPU[9]. With a linear model as inference latency predictor, it is possible to evaluate the accuracy of such a model by computing the correlation coefficient R^2 of the model on CPU and GPU. The coefficients, displayed in figure 5.3, depict the difference in model accuracy between CPU and GPU.

In equation 5.2, it is further assumed that processors use their full capacity, which implies that multi-core processors are modeled as single-core processors with a processing rate equal to the sum of their core processing rates. It is also assumed that the computing resource is shared evenly across threads, *i.e.*, a processor allocates the same proportion of its time to each thread.

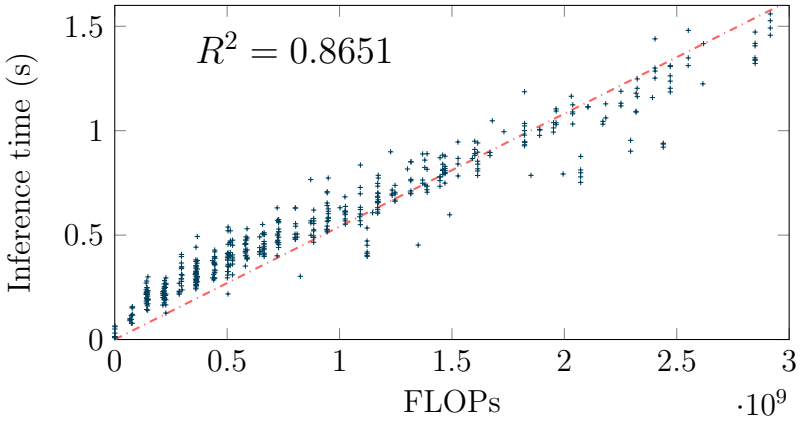
Across all nodes, the thread with highest latency sets the inference throughput for the distributed DNN. Omitting transmission latencies, this implies that once this limiting thread is done computing a data frame, it directly starts processing the

[8]The YOLOv2 model was taken from the ONNX model zoo at https://github.com/onnx/models/

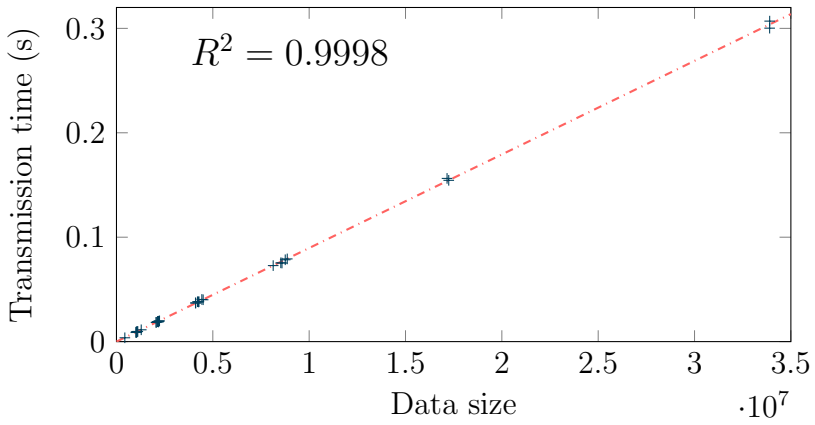
[9]The device used for this experiment is an NVIDIA Jetson Nano



(a) Inference time prediction on CPU



(b) Inference time prediction on GPU



(c) Transmission time prediction

Figure 5.3: Accuracy of linear modeling for inference and transmission latency. The figures show (i) the dependency between number of FLOPs and inference time on an NVIDIA Jetson Nano, running either on CPU (figure 5.3(a)), or GPU (figure 5.3(b)), and (ii) the dependency between transmission times and the size of the intermediary vectors sent between layers in a YOLOv2 model in figure 5.3(c). Each figure displays the correlation coefficient R^2 for a linear predictor with zero value intercept.

next one. Other threads will have idle time before processing the next data frame because their inference latency is lower than this maximum latency, as shown on figure 5.1.

Given a partitioning matrix $\mathbf{P}$, the inference latencies can be expressed as a single vector $\mathbf{T}^c$ of size N where the a-th component is the highest thread inference latency on node N_a, *i.e.*, $\mathbf{T}^c_a = T^c(N_a)$:

$$\mathbf{T}^c = \mathbf{H} \cdot \mathbf{P} \cdot \mathbf{c} \tag{5.3}$$

where $\mathbf{H}$ is the diagonal matrix of inverse node processing rates $diag(\eta_1^{-1}, ..., \eta_N^{-1})$ and $\mathbf{c}$ is the column vector of individual layer consumptions $(c_1, ..., c_L)$.

5.5.3 Transmission latency prediction

The transmission latency can be predicted as follows: the time it takes to send the amount of data $s_{i,j}$ between layers L_i and L_j on edge (N_a, N_b) over a link with measured throughput $\theta_{a,b}$ is $T^t_{i,j}(N_a, N_b) = s_{i,j}/\theta_{a,b}$. The time to achieve data transfers $s_{i,j} \in \mathcal{E}' \subset \mathcal{E}$ over link (N_a, N_b) is:

$$T^t(N_a, N_b) = \theta^{-1}_{a,b} \sum_{s_{i,j} \in \mathcal{E}'} s_{i,j} \tag{5.4}$$

Similarly to the inference latency prediction, this implies that the links are shared evenly between data transfers from N_a to N_b. The transmission latency prediction relies on the estimation of the link throughput between nodes, more precisely the *goodput*, *i.e.*, the amount of useful data transmitted per second.

Similarly to inference latency estimation, this is a first order approximation, which efficiently offers good prediction results, as shown in figure 5.3(c), with a correlation coefficient of $R^2 = 0.9998$. The drawback of this method is its requirement to saturate the links to get a throughput estimation.

Given a partitioning matrix $\mathbf{P}$, the transmission latency matrix $\mathbf{T}^t$ can be defined as a square matrix of size L, with $\mathbf{T}^t_{a,b} = T^t(N_a, N_b)$:

$$\mathbf{T}^t = \left(\mathbf{P} \cdot \mathbf{S} \cdot \mathbf{P}^\top\right) \circ \mathbf{\Theta} \tag{5.5}$$

where $\circ$ is the term by term product, or Hadamard product, $\mathbf{\Theta}$ is the inverse throughput matrix, *i.e.*, a square matrix of size N with $\mathbf{\Theta}_{a,b} = \theta^{-1}_{a,b}$ and $\mathbf{S}$ is the transmission size matrix, *i.e.*, a square matrix of size L with $\mathbf{S}_{i,j} = s_{i,j}$ if L_i sends data to L_j, 0 otherwise.

5.5.4 Optimization objectives

DNN partitioning can be intended to satisfy different use-cases, *e.g.*, minimize delay, energy consumption, cost, or maximize throughput. With the modeling presented in section 5.5, these objectives can be expressed as follows:

End-to-end latency

Minimizing latency is often the principal objective in MEC, and an overall indicator of quality of service in communication networks. In the context of DNN

inference, the end-to-end latency represents the time taken for a frame to traverse the entire network, during one forward pass. This metric can be expressed as the sum of all the terms of both transmission and inference latency matrices:

$$\mathcal{L}\left(\mathbf{P}, \mathcal{G}_A, \mathcal{G}\right) = \sum_{i=1}^{L} \mathbf{T}_i^c + \sum_{a=1}^{N}\sum_{b=1}^{N} \mathbf{T}_{a,b}^t \tag{5.6}$$

Using the matrix notations of sections 5.5.2 and 5.5.3, the optimization problem which find partitionings minimizing the end-to-end inference latency is defined as:

$$\min_{\mathbf{P}} \quad \mathbf{h}^\top \mathbf{P}\mathbf{c} + \mathrm{Tr}\left(\mathbf{P}\mathbf{S}\mathbf{P}^\top\mathbf{\Theta}^\top\right) \tag{5.7}$$

$$\text{s.t.} \quad \mathbf{J}_{1,N} \cdot \mathbf{P} = \mathbf{J}_{1,L} \tag{5.8}$$

$$\mathbf{P}_{a,i} \in \{0,1\} \quad \forall (a,i) \in [\![1,N]\!] \times [\![1,L]\!] \tag{5.9}$$

where $\mathrm{Tr}\left(\cdot\right)$ denotes the trace of a matrix, $\mathbf{h}$ is the column vector of inverse processing rates $(\eta_1^{-1}, ..., \eta_N^{-1})$, $\mathbf{J}_{i,j}$ is the all-ones matrix of size $i \times j$, and conditions (5.14) and (5.15) requiring that each layer be placed on one and only one node.

To solve this partitioning problem, one solution is to use the expression of the latency gradient with regards to $\mathbf{P}$. This can be expressed as:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{P}} = \mathbf{h}\mathbf{c}^\top + \mathbf{\Theta}^\top\mathbf{P}\mathbf{S} + \mathbf{\Theta}\mathbf{P}\mathbf{S}^\top \tag{5.10}$$

Assuming that links are symmetric, this implies that $\mathbf{\Theta}$ is symmetric. If $\mathbf{\Theta}$ is also invertible, this implies that this gradient is equal to zero if and only if:

$$\mathbf{P}\left(\mathbf{S} + \mathbf{S}^\top\right) = -\mathbf{\Theta}^{-1}\mathbf{h}\mathbf{c}^\top \tag{5.11}$$

The diagonal terms of $\left(\mathbf{S} + \mathbf{S}^\top\right)$ being all zeros, this implies that $\det\left(\mathbf{S} + \mathbf{S}^\top\right) = 0$, which means that it is not invertible. The least square method can be applied to solve this linear matrix equation. This method will give a linear approximation of the placement matrix, which can be converted to the closest integer matrix satisfying the conditions of equations 5.7 and 5.9 to obtain a placement matrix which minimizes latency. Using this metric, the DNN partitioning problem can be re-defined to produce placements which minimize latency. However, the solution to this problem is trivial with the modeling presented in section 5.5, *i.e.*, the partitioning which minimizes latency is the one with all computation on the node with the highest processing rate.

Latency is often used in conjunction with other metrics, and optimized jointly, *e.g.*, with cost or energy consumption [95, 96].

Energy consumption

In the fields of MEC and IoT, energy consumption can be an important factor to control. Because of the short battery life of some IoT devices, DNN partitioning can take this factor in consideration when optimizing other metrics such as end-to-end latency.

Energy consumption requires additional modeling than what is presented in section 5.5. The energy consumption caused by a DNN inference workload in the context of DNN partitioning depends on three phases of the device lifetime [95]:

- A switching energy consumption ℓ_i, which corresponds to the energy consumed when turning on the device. This energy consumption is an empirically defined, constant value.

- A run-time energy consumption δ_i, which corresponds to the energy consumed during idle time. This value depends on the idle time spent by the switched on device, and an empirically defined run-time energy consumption rate.

- A computing energy consumption ζ_i, which corresponds to the energy consumed during the computation of the inference workload. This energy consumption depends on the inference time of the partition placed on this device, the size of this workload, and an empirically defined computing energy consumption rate.

Similarly to latency and throughput, this can lead to the definition of a discrete optimization problem, which looks for partitioning schemes which minimize the energy consumption of the whole network.

Inference throughput

The objective of DNN partitioning presented in the remainder of this chapter is to maximize the inference throughput, denoted by C, which corresponds to the inverse of the maximum latency between all the different computation and transmission latencies. Given a partitioning $\mathbf{P}$, the inference throughput is defined as:

$$C(\mathbf{P}, \mathcal{G}_A, \mathcal{G}) = \left(\max_{i,a,b} \left(\mathbf{T}^c_i, \mathbf{T}^t_{a,b} \right) \right)^{-1} \tag{5.12}$$

Maximizing the inference throughput amounts to finding the joint min-max between all the terms of $\mathbf{T}^c$ and $\mathbf{T}^t$. This can be defined as a discrete optimization problem:

$$\min_{\mathbf{P}} \quad \max_{i,a,b} \left(\left(\mathbf{H} \cdot \mathbf{P} \cdot \mathbf{c} \right)_i, \left(\left(\mathbf{P} \cdot \mathbf{S} \cdot \mathbf{P}^\top \right) \circ \mathbf{\Theta} \right)_{a,b} \right) \tag{5.13}$$

$$\text{s.t.} \quad \mathbf{J}_{1,N} \cdot \mathbf{P} = \mathbf{J}_{1,L} \tag{5.14}$$

$$\mathbf{P}_{a,i} \in \{0,1\} \quad \forall (a,i) \in [\![1,N]\!] \times [\![1,L]\!] \tag{5.15}$$

with $\mathbf{J}_{i,j}$ being the all-ones matrix of size $i \times j$, conditions (5.14) and (5.15) require that each layer be placed on one and only one node.

This optimization problem is a Mixed Integer Non-linear Programming (MINLP) problem. The partitioning matrix can be seen as the one-hot encoded version of a vector of size $(1 \times L)$ with values in $[\![1, N]\!]$. MINLP problems are NP-hard, implying that the complexity of finding an optimal DNN partitioning is $\mathcal{O}(N^L)$ since each of the L layers can be placed on N nodes. There are several heuristics to obtain optimal or sub-optimal solutions to this problem in less time than a brute force algorithm, *e.g.*, Genetic Algorithms (GA) [125], Particle Swarm Optimization (PSO) [126], or Branch and Bound (B&B) [127]. The chosen methodology is an adapted B&B implementation, which is presented in section 5.6.1.

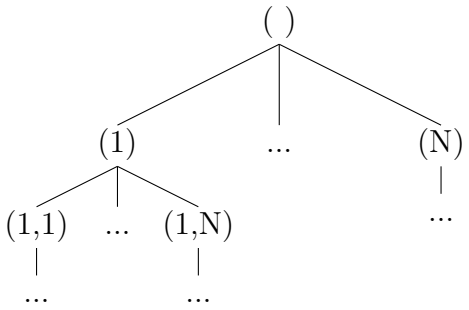


Figure 5.4: Tree representation of the space of all potential DNN partitionings on a physical network of N nodes.

5.6 DNN partitioning solution

This section presents the branch and bound (B&B) adaptation to the DNN partitioning optimization problem defined in section 5.5.4.

5.6.1 Branch and bound algorithm

In B&B algorithms, the solution space is represented as a tree. The process consists of eliminating entire branches in the tree based on the evaluation of the score of the root node. This implies the existence of a tree topology which creates a relationship between the score of a node in the tree and the bounds on the scores of all of its leaves.

For this purpose, a partitioning $\mathbf{P}$ is represented as a list $p = (p_i)_{1 \leq i \leq L}$ with $\forall i \in [\![1, L]\!]$, $p_i \in [\![1, N]\!]$ corresponding to the node assigned to layer i. For example, the partitioning matrix in Equation 5.1 is equivalent to list $p = (1, 1, 1, 2, 2, 1, 1)$, *i.e.*, $\mathbf{P}$ is the one-hot encoded version of p. With this representation, the space of all possible partitionings can be modeled as a tree, as shown in figure 5.4, with each node being a partial version of the full partitioning list. The root node is an empty list, and at every stage $s \leq N$ of the tree, the nodes are all possible lists of size s. The children of a parent node of size s are all lists of size $s + 1$ beginning with the parent node. Partial lists of size s can represent all partitioning lists which start with a given set of s values.

This tree representation of the solution space favors the use of the B&B algorithm in this study. Assuming p_s is a partial list of size s, the children of the node p_s in the tree topology are all lists which start with p_s, *i.e.*, all partitionings where the first s layers are placed according to p_s. The leaf node below p_s corresponding to p_s padded with the last value of p_s is labeled $\tilde{p_s}$. For example, if $L = 5$ and $p_2 = (2, 1)$, then $\tilde{p_2} = (2, 1, 1, 1, 1)$. Given $T^t_{max} = \max \mathbf{T}^t(\tilde{p_s})$ the maximum value of the transmission latency matrix for partitioning $\tilde{p_s}$, all leaf nodes p below p_s will have a higher maximum transmission latency than $\tilde{p_s}$, *i.e.*, $\forall p, \max \mathbf{T}^t(p) \geq T^t_{max}$. This is explained by the fact that given a partial partitioning, any displacement in the other layers will only add data transfers between nodes.

During the process of looking for an optimal partitioning p, which maximizes the inference throughput $C(p, \mathcal{G}_A, \mathcal{G}_N)$, and given a current best achievable throughput best_throughput, the B&B optimization consists of eliminating entire branches of the tree representing the solution space by evaluating transmission times in partial

partitionings. This process allows the B&B optimization to quickly eliminate numerous cases compared to a brute force algorithm. For example, if the throughput between nodes is low compared to the compute capacity, *e.g.*, with nodes connected through a low throughput wireless connection, the optimal solution is often to keep all computation on a single node. The advantage of the B&B algorithm is that it terminates after N operations in such simple cases by eliminating all partial partitionings in the first stage of the tree. Conversely, in cases where links have higher throughputs, the complexity of the B&B can be higher since it will be unable to eliminate large branches.

5.6.2 Complexity

The overhead caused by the total number of partitions in a real world implementation is neglected in this approach. This added latency is correlated with the total number of partitions. With each partition, the data transfer from NIC to memory causes an additional latency, related to the PCIe throughput, memory throughput and size of the transferred data. Assuming that the data transfer throughput is limited by the memory throughput, a simplistic evaluation of this latency would change the computation latency in equation 5.2 to:

$$T^c(N_p) = \nu^{-1} \sum_{(i,j)\in\mathcal{C}} s_{i,j} + \eta_p^{-1} \sum_{L_i\in\mathcal{L}'} c_i$$

In this expression, ν is the memory throughput, *i.e.*, the read/write speed to and from the memory, and $(i,j) \in \mathcal{C}$ are all the layers L_i and L_j such that edge (L_i, L_j) is a split point with associated data transfer $s_{i,j}$.

Early stopping

With an increasing number of partitions, both the performance and inference latency prediction accuracy will decrease. For this reason, and for the increase of complexity in in cases with high available network throughput, an early stopping mechanism is added to limit the number of split points in the final partitioning, *i.e.*, B&B will only explore partial partitioning with a maximum of S split points, further limiting the size of the explored tree. The complete B&B algorithm with limited split points is shown in algorithm 2.

Limiting the number of split points lowers the complexity of this MINLP problem. For N the number of nodes, and L the number of DNN layers, instead of $\mathcal{O}(N^L)$ for a brute force algorithm, the number of possible partitioning schemes considered by the algorithm is now lowered to:

$$\sum_{k=0}^{S} \binom{L-1}{k} N\,(N-1)^k \tag{5.16}$$

5.7 Comparison with standard MINLP solvers

To confirm the selection of B&B as a solver for the DNN partitioning problem, this section compares the performance of B&B with two other solvers for MINLP problems: the Genetics Algorithm (GA) [125], and Particle Swarm Optimization

Algorithm 2 Branch and Bound Algorithm

Input: $\mathcal{G}_A, \mathcal{G}_N, S$

Output: best_partitioning, best_throughput

```
queue ← {[ ]}
best_throughput ← C([1, ..., 1], G_A, G)
best_partitioning ← [1, ..., 1]
while queue is not empty do
  get p from queue
  generate all children p ‖ {i}, ∀i ∈ ⟦1, N⟧
  compute t_i the maximum transmission time T^t(p ‖ {i})
  for i = 1 to N do
    if t_i > best_throughput^-1 then
      discard p ‖ {i}
    else if t_i ≤ best_throughput^-1 then
      if S(p ‖ {i}) < S then
        add p ‖ {i} to queue
      end if
      if C(p ‖ {i}, G_A, G_N) > best_throughput then
        best_throughput ← C(p ‖ {i}, G_A, G)
        best_partitioning ← p ‖ {i}
      end if
    end if
  end for
end while
```

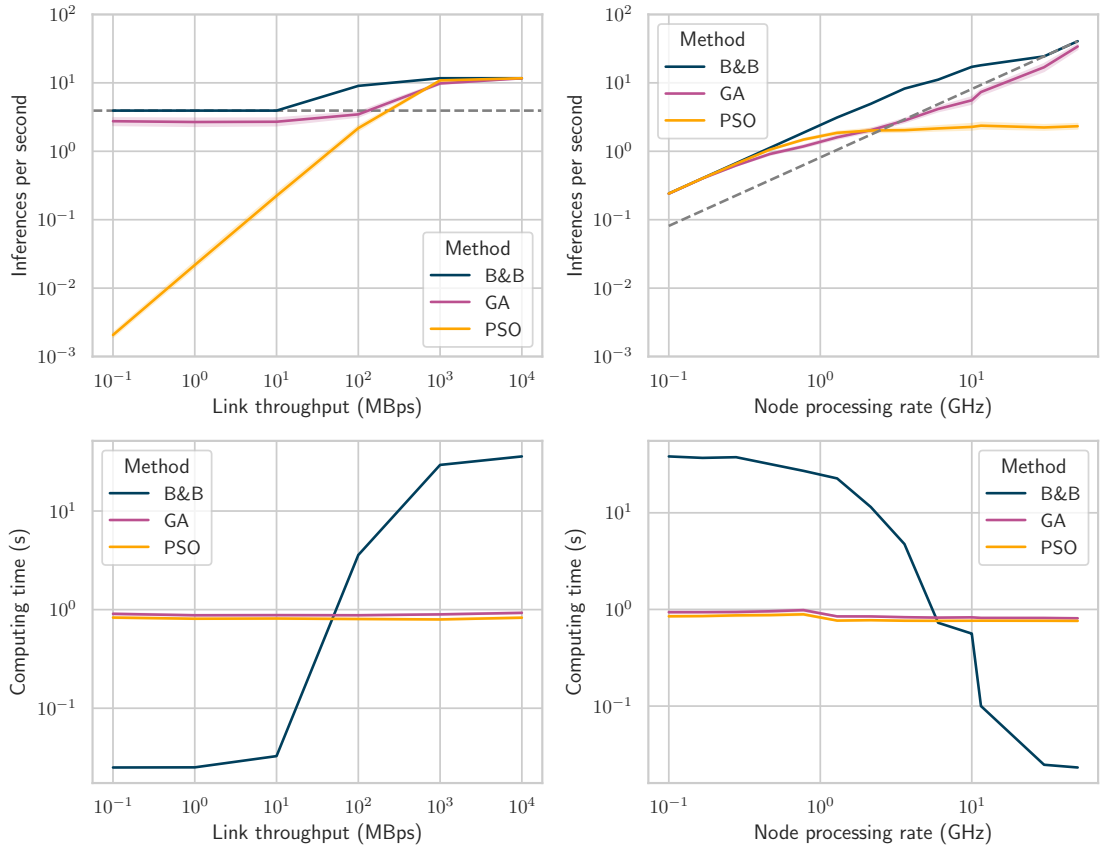


Figure 5.5: Comparative analysis of Branch and Bound (B&B), Genetics Algorithm (GA), and Particle Swarm Optimization (PSO), on the DNN partitioning problem to maximize the inference throughput.

(PSO) [126]. The methods are evaluated on the achieved partitioning performances and the complexity to reach these solutions.

Because the space of all potential solutions is too wide to sample fully, this evaluation compares the solvers by isolating parameter influences, and looking at their variations one at a time. Each of the four observed parameters has a default value, and each figure presents how they each influence the solvers performance and complexity. The four chosen parameters are (i) the number of nodes (default is $N = 4$), (ii) the partitioned DNN model (default is YOLOv2 [124]), (iii) the link throughput (default is 10MBps which corresponds to a 802.11 connection), and (iv) the node processing rate, default is 5GHz which corresponds to a Raspberry Pi 4[10]. The results are shown in figures 5.5 and 5.6.

The figures show that the inference throughput achieved by B&B is always greater or equal to the unpartitioned inference throughput, with varying levels of complexity. GA and PSO show the opposite behaviour, *i.e.*, the complexity remains bounded (both methods run for a fixed number of iterations), but the achieved solutions are often either sub-optimal, or worst than the unpartitioned solution.

In the case of "simple" partitionings, *i.e.*, cases where no partitioning can improve the unpartitioned inference cadence, the figures show that B&B can reach this

[10] https://www.raspberrypi.com/products/raspberry-pi-4-model-b

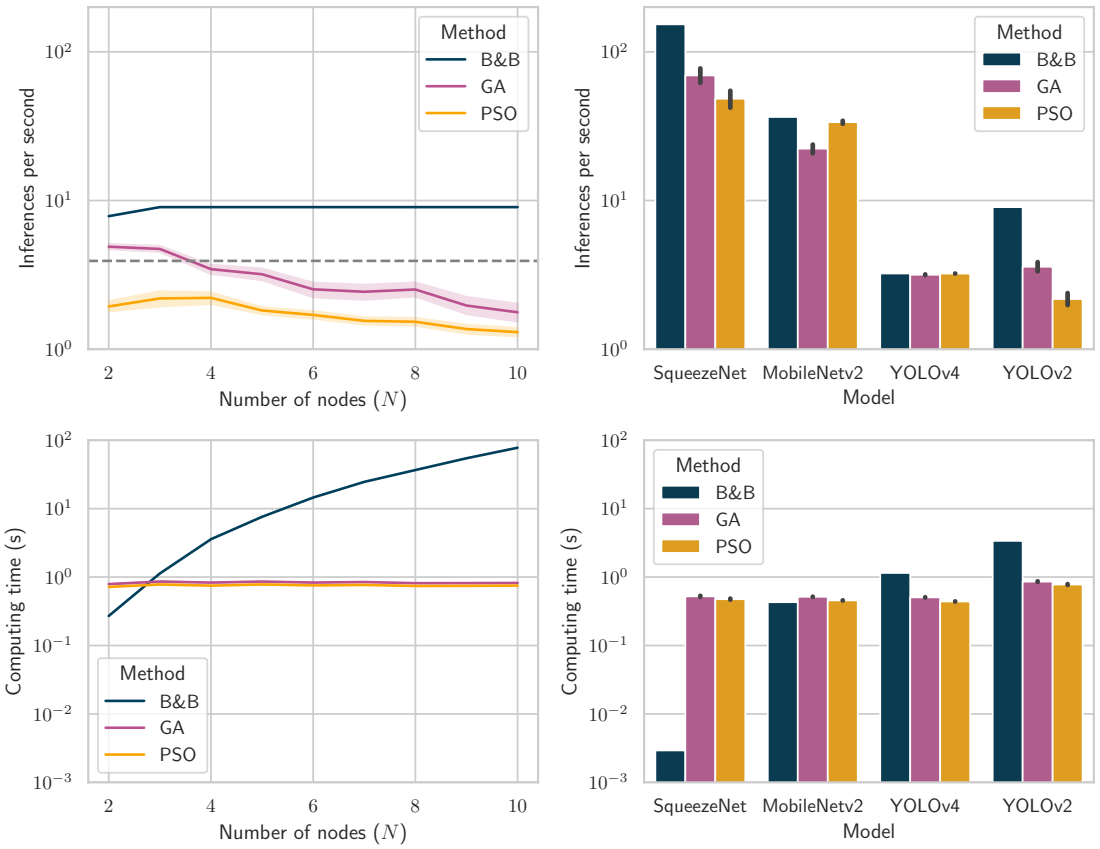


Figure 5.6: Comparative analysis of Branch and Bound (B&B), Genetics Algorithm (GA), and Particle Swarm Optimization (PSO), on the DNN partitioning problem to maximize the inference throughput.

conclusion with a very low complexity, while GA and PSO fail to find the optimal solution.

Although cases with a high complexity can cause important computation times, these evaluations confirm the choice of B&B as a solver for the DNN partitioning use-case, since it does not require real-time partitioning decision, and can afford to run this optimization once before deployment, assuming stable network conditions.

5.8 Summary of results

Deploying DL applications in the cloud is convenient because it allows on-demand easy access to computing resources. However, latency or privacy sensitive applications may not be able to exchange data and models with the cloud, while still requiring the same inference throughput to run with good performance. In such cases, DNN partitioning can offer a complementary alternative to hardware acceleration and model optimization to increase inference throughput. This chapter has described such an approach to DNN partitioning, which extends previous works by allowing for multiple split points and multiple threads. This enables a better distribution of the workload, depending on the DNN and network properties.

In this context, this chapter has presented general assumptions to represent the problem and predict the inference throughput. During this process, this chapter has quantified the limitations of inference and transmission latency prediction in edge environments, and what it implies when modeling DNN partitioning. With these assumptions, the problem is defined as an optimization process, with the objective of maximizing the overall inference throughput. This problem is a MINLP problem, and therefore NP-hard.

This chapter has introduced a B&B solver, and compared it to other standard solvers for MINLP problem, namely GA and PSO, to illustrate that B&B (i) always finds better solutions than its alternatives, and that (ii) it reaches simple solutions quickly, *e.g.*, when no partitioning can improve the unpartitioned inference throughput.

Chapter 6

Performance and complexity analysis

Although the B&B method for DNN partitioning presented in chapter 5 can lead to inference throughput accelerations, the influence of the DNN, and of network properties on the performance and complexity of the solutions remains to be determined. Depending on the values of link throughput, processing rate, DNN and network properties, the B&B algorithm can lead to insufficient levels of performance improvements, and computation times which are too long in some use-cases. In a given context, understanding these relationships prior to the deployment can be essential, because it allows for proper sizing of the network equipment to satisfy service level objectives. For a given DNN, this can determine which hardware to deploy, in order to satisfy service level objectives. Similarly, for given hardware and link properties of the edge network, this can influence the choice of models, which will benefit most from DNN partitioning. This understanding is particularly important at the edge, where hardware is subject to form factor, energy consumption, and performance requirements.

The key performance indicators studied in this chapter are the following:

- *The conditions leading to a performance improvement.* The optimal solution in the DNN partitioning problem can be to simply keep all computation on a single node, because any split in the computation will introduce transmission times too important, to accelerate inference throughput. This defines the conditions under which DNN partitioning can be beneficial, and is important to understand before dimensioning a network.

- *The expected performance improvement.* Similarly, some use-cases can require a minimal inference throughput, to guarantee application service level objectives, *e.g.*, in the case of object detection and tracking in video streams, where low inference throughputs can lead to loss of information [89].

- *The expected time to reach an optimal solution.* The complexity of the B&B algorithm depends on (i) the parameters of the algorithm, (ii) the size of the DNN, and (iii) the size of the network. Applications requiring a bounded B&B computation time need to understand the influence of these parameters on the algorithm complexity and the implications on the achieved performance.

6.1 Statement of purpose

The purpose of this chapter is to study the DNN partitioning solution presented in chapter 5 to identify the trade-off between performance and complexity, under various DNN and network conditions. This chapter provides deterministic bounds on achieved performance and complexity, to assist deployment of DNN partitioning solutions. This chapter also presents simulations in edge computing environments, as well as experiments in real edge deployments to confirm the simulated results.

The contributions of this chapter are the following:

- Simulations to explore the possible performance improvements with varying network and DNN properties.
- Identification of deterministic regions in the network and DNN properties leading to the existence of optimal partitionings and the cost to compute such solutions, *i.e.*, the conditions under which DNN partitioning is beneficial.
- Experimental results, confirming the regions defined in the simulations, as well as the prediction accuracy and final inference throughput acceleration for homogeneous and heterogeneous environments.

6.2 Chapter outline

Section 6.3 presents simulation results to quantify the performance and complexity of the B&B algorithm, as well as conditions leading to the existence of a partitioning which improves the inference throughput in a homogeneous network. Section 6.4 presents experimental results confirming the simulations of section 6.3, and shows inference throughput improvements on a heterogeneous experimental set-up. Finally, section 6.5 discusses the results and expands on the broader applicability of this method.

6.3 Simulations

This section presents simulations to evaluate the impact of the algorithm parameters on the B&B algorithm performance and complexity. The worst case complexity for B&B is described in equation 5.16, when all of the transmission times computed in the partial partitionings exceed the best achieved time (cf. section 5.6), but the effective B&B complexity varies according to the DNN and network properties. The following variables are explored:

- The relationship between number of nodes N and the maximum allowed number of split points S, which determines the required number of B&B iterations to reach the optimal solution.
- The relationship between link throughput θ and node processing rate η, which determines the existence of a partitioning which improves the inference throughput.

These interactions are evaluated via the following metrics:

- The inference throughput, measured in inferences per second.
- The number of B&B iterations needed to reach the solutions, *i.e.*, the number of explored partitionings in the solutions tree. This metric is used in place of computation time, to abstract away the specific device performance.[1]

To isolate the contribution of each variable, simulations are run in a homogeneous network scenario, *i.e.*, where all nodes have identical processing rates, and all links have identical throughputs.

6.3.1 Number of nodes and split points

The set-up consists of a homogeneous network with processing rates set at 5GHz (effective processing rate of a standard edge device, *e.g.*, a Raspberry Pi 4[2]), and link throughputs at 10MBps (effective throughputs for connections through 802.11). The number of compute nodes N on the network varies between 1 and 6, and the maximum number of split points S from 0 (keeping all computation on a single node) to 5 (for a total of 6 partitions across the network).

The results in figure 6.1 show that (figure 6.1(a)), for $N > 2$ nodes in the network, B&B finds partitionings which increase inference throughput by 1.9× to 2.3× the throughput of the unpartitioned solution ($S = 0$). By increasing the maximum number of allowed splits, S, the algorithm is able to find better partitionings and improve the inference throughput, at the expense of additional complexity, depicted in figure 6.1(b).

Figure 6.1(a) also illustrates that the best achievable throughput does not improve above $S > 3$. This is an argument for limiting the B&B maximum split point value S. Since the performance stays identical while the complexity increases significantly, choosing $S = 3$ under these conditions both maintains a reasonable complexity and reaches the optimal solution. In this case, adding a node to the network, or the possibility of another split point, will not lead to a better partitioning. In a homogeneous network, this implies that any relocation of layers to another node will incur transmission latencies higher than the gain in computing latency of the current partitioning. This leads to a bound on the value of S, after which point the optimal partitioning is the best possible achievable partitioning. This bound can be expressed as:

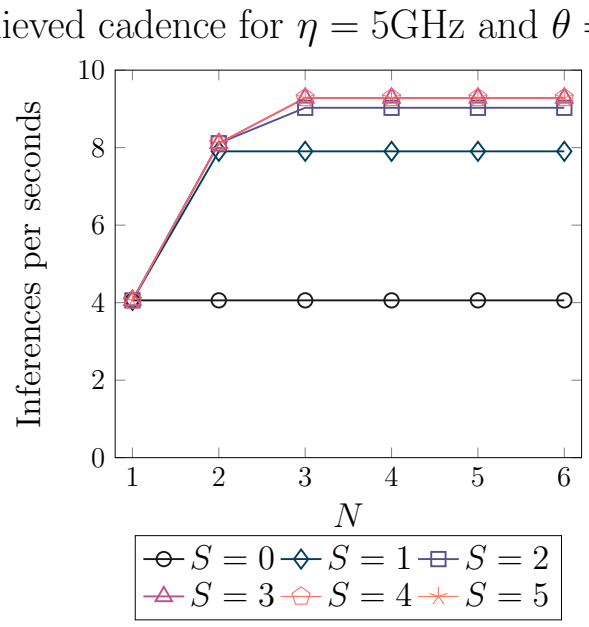
$$S = \frac{T^c_{\text{mono}}}{T^t_{\text{disp}}} < \frac{\theta \sum_{L_i \in \mathcal{L}} c_i}{s_{\min} \eta} \tag{6.1}$$

with T^c_{mono} the computing latency when keeping all computation on a single node, T^t_{disp} the transmission latency caused by a layer displacement η, and θ the node processing rate and link throughput values in the homogeneous network, $\sum_{L_i \in \mathcal{L}} c_i$ is the sum of all layer consumptions in the DNN, and $s_{\min}$ is the minimal inter-layer data transfer size. This expression can be used to define an upper bound on the value of S, and limit the B&B computation time.

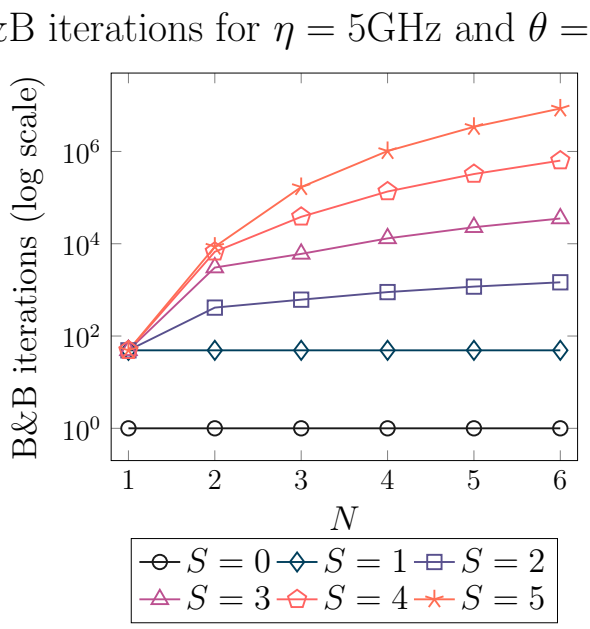
For the remainder of this paper, experiments and simulations will be run with a maximum number of splits $S = 3$ to limit the complexity of the algorithm, with

[1] For reference, when running B&B on a desktop computer with an Intel Core i7 processor, it took 17 milliseconds to run through 10^2 iterations, 1.269 seconds to run through 10^4 iterations, and 5 minutes to run through 10^6 iterations.

[2] https://www.raspberrypi.com/products/raspberry-pi-4-model-b



(a) B&B achieved cadence



(b) B&B complexity

Figure 6.1: Impact of the number of nodes N and the maximum number of split points S on B&B achieved inference throughput (figure 6.1(a)), and complexity (figure 6.1(b)), for a YOLOv2 model on a homogeneous network.

near optimal partitioning in the described set-up, which corresponds to a typical edge scenario.

6.3.2 Processing rate and link throughput

The simulated environment consists of a standard implementation of YOLOv2 [124], deployed on a network with $N = 4$ compute nodes. B&B is run with a maximum number of split points set to $S = 3$, for node processing rates between 0.1GHz and 100GHz, and link throughputs between 1MBps and 10GBps. The ranges in value for processing rate and link throughput were chosen to cover a wide spectrum of edge scenarios:

- Processing rates between 0.1GHz and 100GHz cover CPUs of systems on a chip, *e.g.*, the Qualcomm Snapdragon suite[3] with processing rates between several 500MHz and 1GHz, and specialized AI embedded systems, *e.g.*, the NVIDIA Jetson TX2 module[4] measured at an equivalent processing rate of 30.7 GHz in the experiments of section 6.4.

- Link throughputs between 1MBps and 10GBps correspond to link throughputs covering 802.11g connections at several MBps, and Gigabit Ethernet links.

Figure 6.2 displays the impact of link throughput and node processing rate, on both the B&B achieved inference throughput (in inferences per second) and complexity (in number of partitionings evaluated by B&B in the simulation). Level lines are included to facilitate interpretation of the figures. Darker colors on the figures correspond to lower inference throughputs (respectively B&B iterations) and lighter colors correspond to higher values, for a set-up of devices with the corresponding processing rate and link bandwidth value.

Results show that achieved inference throughput and B&B iterations increase with both processing rate and link throughput. This is expected, since faster processors yield faster inferences, and better link throughput creates possibilities for improved partitionings.

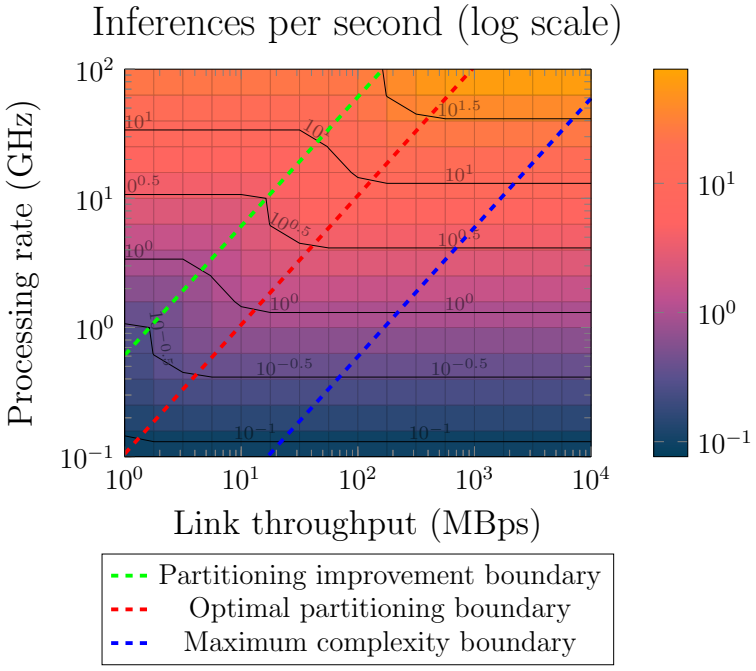
Additionally, these simulations highlight the existence of distinguishable regions in the two heatmaps of figure 6.2, separated by boundaries, corresponding to fixed link throughput to processing rate ratios: the partitioning improvement boundary, the optimal partitioning boundary and the maximum complexity boundary, displayed by the three dotted lines of figure 6.2. These boundaries are described in the following.

Partitioning improvement boundary

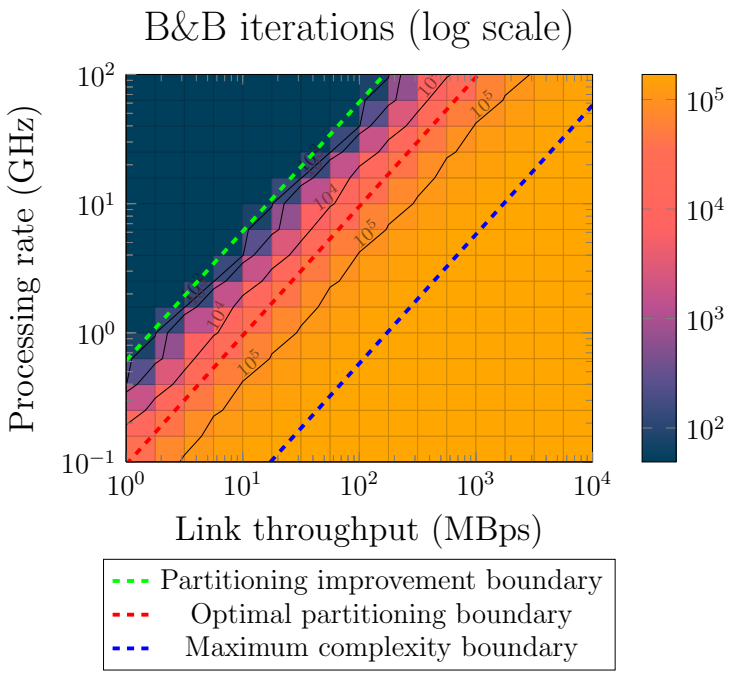
The green dotted line separates the top-left region of figures 6.2(a) and 6.2(b), where the optimal solution consists of keeping all the computation on the same node, and the bottom-right region, where a non-trivial partitioning that improves inference throughput exists. This boundary corresponds to conditions on the ratio between link throughput and node processing rate under which B&B starts to explore improved solutions in algorithm 2: $\max \mathbf{T}^t \leq \text{best\_throughput}^{-1}$. This is

[3] https://www.qualcomm.com/snapdragon

[4] https://developer.nvidia.com/embedded/jetson-tx2



(a) B&B achieved inference throughput



(b) B&B complexity

Figure 6.2: Impact of node processing rate and link throughput on B&B achieved inference throughput (figure 6.2(a)) and complexity (figure 6.2(b)), for a YOLOv2 model on a homogeneous network.

validated under the condition that the smallest data transfer latency between nodes exceeds the unpartitioned computing latency, *i.e.*, the existence of partitionings which improve the unpartitioned inference throughput is subject to:

$$\frac{s_{\min}}{\theta} < \frac{\sum_{L_i \in \mathcal{L}} c_i}{\eta} \qquad i.e., \quad \frac{\theta}{\eta} > \frac{s_{\min}}{\sum_{L_i \in \mathcal{L}} c_i} \tag{6.2}$$

with η and θ the node processing rate and link throughput values in the homogeneous network, $\sum_{L_i \in \mathcal{L}} c_i$ the sum of all layer consumptions in the DNN, and $s_{\min}$ the minimal inter-layer data transfer size. This expression is a particular case of equation 6.1, with a number of split-points $S = 1$.

This defines a criterion on the region where distributing inference can improve the overall performance: in a homogeneous scenario, for every DNN, the link throughput to node processing rate ratio $\frac{\theta}{\eta}$ (which is a property of the network) needs to exceed a deterministic value described in equation 6.2 (which only depends on properties of the DNN). This boundary is represented by the green dotted line in figure 6.2. For example, this corresponds to $\frac{\theta}{\eta} \approx 1.64 \times 10^{-3}$ for the YOLOv2 implementation used in the simulations.

Optimal partitioning boundary

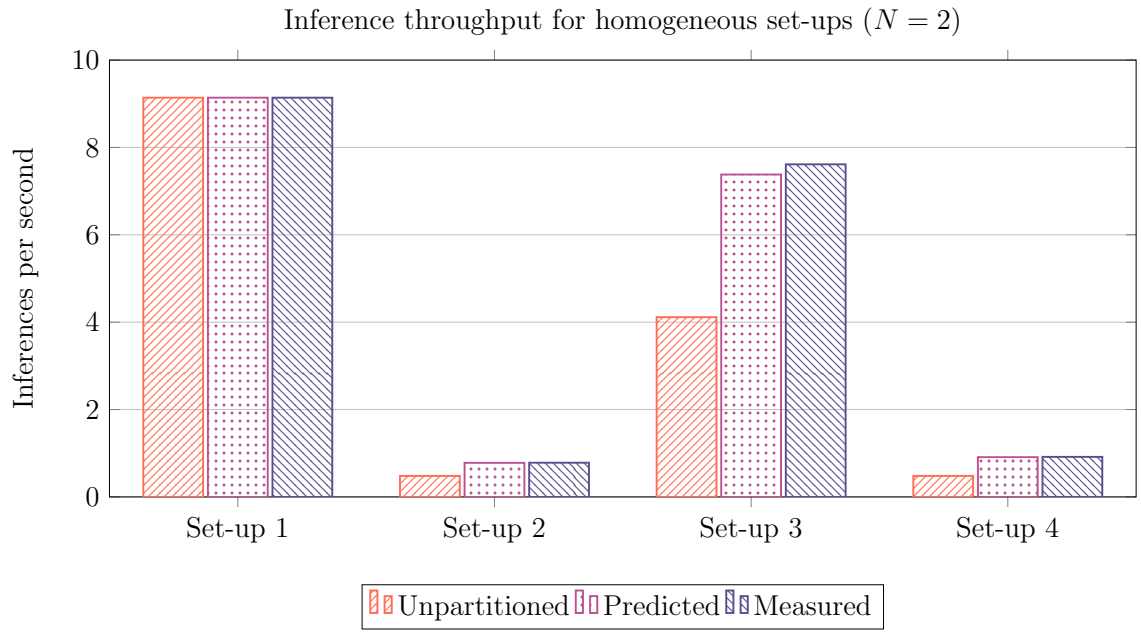
The red dotted line in figure 6.2 corresponds to $\frac{\theta}{\eta} \approx 9.52 \times 10^{-3}$ and delimits the point of diminishing returns, *i.e.*, the maximum achievable inference throughput for YOLOv2. For higher values of link throughput to processing rate ratio $\frac{\theta}{\eta}$ (bottom right), the inference throughput remains identical while the number of evaluated partitionings by B&B continues to increase. The additional evaluated partitionings have lower inference throughputs than the optimal solution. This explains why the level lines in figure 6.2(a) are horizontal in that region, since the inference throughput only depends on the processing rate. Increasing the ratio above this boundary (*e.g.*, by increasing link throughputs) does not yield better solutions, and increases complexity.

Maximum complexity boundary

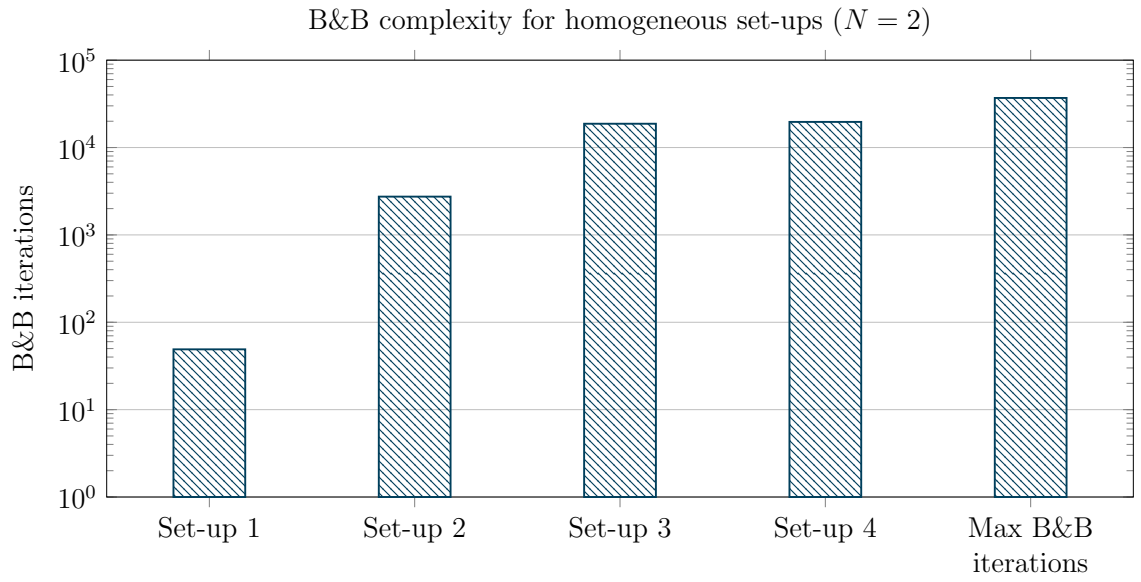
The blue dotted line in figure 6.2 corresponds to conditions under which the number of evaluated partitionings by B&B is maximal, and corresponds to $\frac{\theta}{\eta} \approx 0.168$ in the simulated environment. For higher values of the ratio, *i.e.*, on the bottom right part of this boundary, the complexity of B&B has reached a maximum and remains constant. Notably, this maximum number of partitionings evaluated by B&B is around 1.7×10^5. This measured maximum complexity remains below the theoretical maximal value (around 1.9×10^6 iterations in this context, cf. equation 5.16), and orders of magnitude below the brute force algorithm complexity, which would need to evaluate $N^L \approx 10^{29}$ partitionings (for the given YOLOv2 implementation).

6.3.3 Discussion

These simulations allow to identify three boundaries, which delimit the conditions under which a partitioning can improve the unpartitioned inference throughput, in a homogeneous network, and depend on the DNN, and on the network prop-



(a) Inference throughput



(b) B&B iterations

Figure 6.3: Achieved inference throughput and complexity for a YOLOv2 model in homogeneous experimental set-ups (table 6.1). Figure 6.3(a) compares the achieved inference throughput with the unpartitioned throughput, and the B&B predicted throughput. Figure 6.3(b) compares the effective number of iterations required to compute the partitioning with the maximum number of iterations for $S = 3$ split points.

Set-up	Processor	Link
1	NVIDIA Pascal GPU (256 CUDA cores)	Wi-Fi
2	Quad-core ARM A57 CPU	Wi-Fi
3	NVIDIA Maxwell GPU (128 CUDA cores)	Ethernet
4	Quad-core ARM A57 CPU	Ethernet

Set-up	Processing rate (η)	Link throughput (θ)	Ratio (α)	B&B time
1	30.7 GHz	10 MBps	3.26×10^{-4}	4 ms
2	1.57 GHz	10 MBps	6.37×10^{-3}	165 ms
3	14.7 GHz	1.4 GBps	9.52×10^{-2}	2.231 s
4	1.57 GHz	1.4 GBps	9.82×10^{-1}	2.494 s

Table 6.1: Experimental set-ups with corresponding link throughput to node processing rate ratios, and associated B&B computation times. Each experimental set-up corresponds to properties in one of the four separate zones identified in figure 6.2.

erties The partitioning improvement criterion defines these conditions on the link throughput to processing rate ratio, and can be anticipated prior to the deployment.

While the optimal partitioning boundary and maximum complexity boundary depend on the number of nodes N, and on the maximum number of split points S, the location of the partitioning improvement boundary is independent of these variables, or the number of layers. DNN partitioning is independent of the size of the network and the DNN, but depends only on the relationship between (i) $\frac{\theta}{\eta}$ (a property of the network), and (ii) $\frac{s_{min}}{\sum_{L_i \in \mathcal{L}} c_i}$ (a property of the DNN).

Regarding complexity, it has to be noted that the optimization process is a one-time operation that decides on a partitioning which will remain relevant for long periods, *i.e.*, longer than the order of magnitude of B&B computation times depicted in figure 6.1. The necessity to recompute a partitioning would only arise if the system experiences persistent changes in the node processing rates or link throughputs. In "healthy" network scenarios, where faults, or events causing persistent changes, are rare, this paper argues that partitioning computation times of B&B can fit use-cases with one-time deployments.

In more constrained use-cases, the maximum number of split points S can further act as a tuning parameter for the optimization complexity, *e.g.*, if the computation of an optimal partitioning is more frequent, at the expense of the achieved inference throughput.

B&B can reach optimal solutions with unlimited split points ($S \geq L-1$), but the time to reach these solutions increases with α. Nevertheless, it is possible to observe from figure 6.1(a) that this solution can lead to higher inference throughputs than methods limited to a single point ($S = 1$).

6.4 Experiments

This section presents experimental results, evaluating the accuracy of the model. Section 6.4.1 describes experiments in homogeneous scenarios with two identical nodes, performed to test the validity the identified boundaries in the simulations

presented in section 6.3. Section 6.4.2 describes experiments and presents results in networks with heterogeneous nodes.

6.4.1 Homogeneous network

Table 6.1 details links, processing rates, and corresponding ratios, of the scenarios tested.

- Set-up 1 corresponds to conditions above the partitioning improvement boundary in figure 6.2 (section 6.3.2), *i.e.*, no partitioning can improve the inference throughput (*i.e.*, the set-up does not satisfy the partitioning improvement criterion of equation 6.2).

- Set-up 2 corresponds to conditions between the partitioning improvement boundary and the optimal partitioning boundary (section 6.3.2), *i.e.*, partitionings which improve the inference throughput exist.

- Set-up 3 corresponds to conditions between the optimal partitioning boundary and the maximum complexity boundary (section 6.3.2), *i.e.*, B&B is expected to find the best possible partitioning.

- Set-up 4 corresponds to conditions below the maximum complexity boundary, *i.e.*, the partitioning is the best possible solution, and the number of B&B iterations is maximal.

The values in table 6.1 are experimental and were measured by benchmarking devices and links. Partitions are deployed and run for a YOLOv2 model and a maximum number of splits $S = 3$, for each of the experimental set-ups. The inference is run on a 1280x720 webcam video stream, with 30 frames available per second.

Figure 6.3 displays the inference throughputs, and compares them to the B&B predicted inference throughputs (cf. section 5.5), and to the baseline, defined as the throughput achieved by keeping all of the computation on a single node. The results correspond to the simulations and confirm the existence of the four identified regions of figure 6.2.

6.4.2 Heterogeneous network

This section presents an experiment which considers the case of adding a single device with varying capacities to a set-up with a single device. This experiment aims to both show results on simple heterogeneous set-ups, and to illustrate the case of an additional device being added to a network to improve the overall performance, *e.g.*, adding a processor in proximity to a smart camera to increase its inference throughput.

The node which remains unchanged in this experiment[5] has a processing rate η_0=14.7GHz, and figure 6.4 shows the measured inference throughputs when adding devices with varying processing rates to the network, with two different link throughput values, compared with their predicted values. The figure also depicts bounds on possible solutions: the lower bound is the unpartitioned inference throughput,

[5]The node processor is an NVIDIA Maxwell GPU (128 CUDA cores).

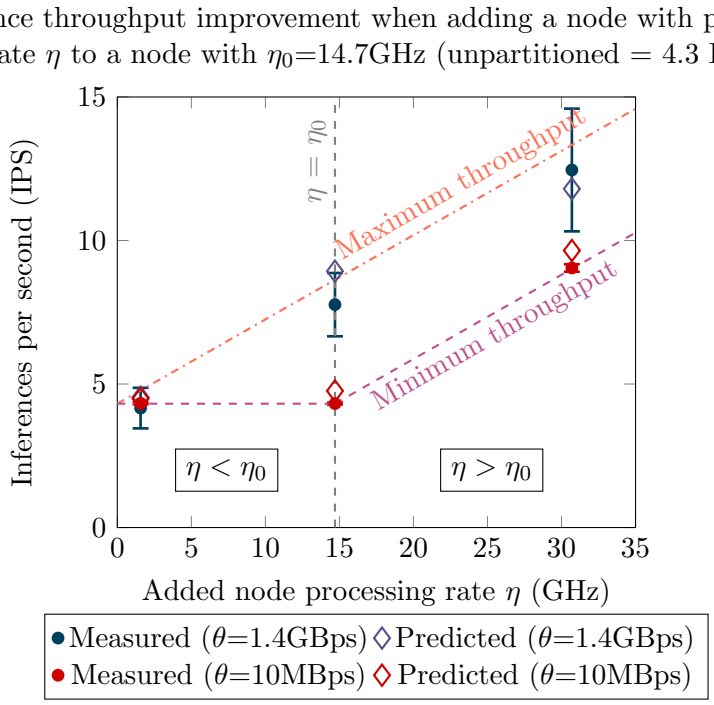


Figure 6.4: Inference throughput improvement when adding devices with varying processing rates η to a device with a processing rate of η_0=14.7GHz. The figure shows measured throughput values with their standard deviation and compares them with the predicted inference throughputs.

i.e., the throughput when placing all computation on the fastest node, and the upper bound corresponds to a device with the sum of the two processing rates, *i.e.*, corresponding to a perfectly even distribution of the inference workload.

This experiment shows that under good link conditions, *i.e.*, above the partitioning improvement boundary (section 6.2), the inference throughput can be close to the maximum possible throughput. Notably, the expression of the partitioning improvement boundary differs from its expression in the homogeneous case (defined in the same way as equation 6.2):

$$\frac{s_{\min}}{\theta} < \frac{\sum_{L_i \in \mathcal{L}} c_i}{\eta_{\max}} \qquad i.e., \quad \theta > \frac{s_{\min}\eta_{\max}}{\sum_{L_i \in \mathcal{L}} c_i} \tag{6.3}$$

for a heterogeneous case with $N = 2$ nodes, with $\eta_{\max}$ being the maximum processing rate between the two nodes.

This means that for link throughputs below the value $\frac{s_{\min}\eta_{\max}}{\sum_{L_i \in \mathcal{L}} c_i}$, optimal partitioning consist in keeping all layers on the fastest node, *i.e.*, following the minimum throughput line, as is the case when the nodes are connected via a Wi-Fi connection with θ=10MBps in figure 6.4. For higher link throughputs, there exist partitionings which improve the inference throughput, and inference throughput values are higher than the lower bound in figure 6.4, as is the case when the nodes are connected via Ethernet links with θ=1.4GBps.

6.5 Results, scope and limitations

This section discusses the scope and limitation of the simulations and experiments on DNN partitioning.

6.5.1 Conditions for homogeneous networks

Simulations (section 6.3) and experiments (section 6.4), on homogeneous networks, have identified conditions, bounds, and closed expressions, for performance and complexity of DNN partitionings. Under *homogeneity* (*i.e.*, nodes and links in the underlying network having equivalent capability), these expressions allow to dimension the underlying network, and to predict the partitioning outcome.

- *The partitioning improvement boundary* (section 6.3.2) describes the conditions under which there are partitionings that improve the monolithic inference throughput, *i.e.*, the throughput when placing all computation on a single node. This is extended to the heterogeneous case in equation 6.3, and allows to estimate the values of link bandwidth and node processing rate, necessary to achieve an inference throughput improvement.

- *The upper bound on the number of split-points* necessary for an optimal solution (equation 6.1) is the number of split-points above which the complexity increases while throughput remains constant. This allows to minimize the B&B computing time, while accessing maximal inference throughput.

- *The maximal B&B complexity*, with a chosen maximum number of split-points (equation 5.16), corresponding to the maximal number of partitionings evaluated by B&B.

These results allow to derive, under conditions of perfectly even distribution of workload over homogeneous nodes and links, an upper bound on the inference throughput of the partitioning strategy:

$$C(\mathbf{P}, \mathcal{G}_A, \mathcal{G}) < \min\left(\frac{\eta \min(N, S)}{\sum_{L_i \in \mathcal{L}} c_i}, \frac{\theta}{s_{\min}}\right) \tag{6.4}$$

These expressions are valid on a homogeneous network. For non-homogeneous networks, the derivation of similar performance and complexity bounds is, of course, specific to the characteristics of the network.

6.5.2 Scope and limitations

This section discusses the limitations of this work, and the presented assumptions, to illustrate their scope of applicability.

Input data

This study has presented DNN partitioning through video stream data applications. Although its applicability has not been illustrated with other data types, there are no assumptions in the modeling restraining this method from applying to other data types, *e.g.*, audio, text, or telemetry data. The only limiting assumption

in this study is that the model used a feed-forward DNN, with data of constant size across requests.

Optimization problem definition

This study has covered use-cases which require a maximal inference throughput, omitting optimization objectives included in the related work of section 5.1, *e.g.*, latency, energy consumption, cost, or a combination of these previous metrics. However, the presented assumptions and modeling can be exploited to describe other use-cases. For example, DNN partitioning can cover contexts which:

- jointly optimize several metrics, *e.g.*, throughput, latency, energy consumption, monetary cost, drop rate, node up-time, link usage, etc.
- dynamically adapt what metric to optimize depending on the received data. For example, DNN partitioning application on video streams can choose to optimize throughput to avoid missing detections, and switch to latency when an event occurs in the system, to enable low response times.
- add other constraints to the MINLP problem, *e.g.*, a minimal number of split points, a partial node to layer mapping, *e.g.*, in order to keep sensitive computation on dedicated nodes.

All of these assumptions can be expressed as optimization objectives, or constraints in the discrete optimization problem of section 5.6.

Limits of experimental results

The experiments presented were designed to confirm the boundaries, identified in the simulations of section 6.3, and to illustrate performance in a simple heterogeneous use-case. This study also has assumed that the network properties remain fixed over time.

Completely exploring the influence of other parameters on the DNN partitioning performance and complexity requires additional experiments, following the pattern and methodology of those presented in this chapter. These parameters include the number of compute nodes and number of split points in large networks, the complexity of DNN structures, the heterogeneity of layer consumptions and data transfer sizes, or the heterogeneity of link bandwidths and processing rates in large networks. Additional exploration is required to provide a better understanding of how the system behaves when (i) other processes are dynamically allocated to compute nodes, (ii) links are dynamically used by other processes, or (iii) faults occur on the system (*e.g.*, node failure, routing change, packet drops).

6.6 Summary of results

This chapter has introduced a theoretical analysis of the complexity and inference throughput results of the branch and bound algorithm, proposed in chapter 5

This analysis has led to identification of the partitioning improvement boundary, deterministic conditions on the network and DNN properties leading to a performance improvement through partitioning. This chapter has further allowed to

quantify the cost to compute such solutions, and their expected performance, in a homogeneous network context. This result defines the scope of validity of DNN partitioning in edge computing environments, and only depends on (i) the DNN data transfer size to layer consumption ratio, and (ii) the link throughput to processing rate ratio of the underlying network.

The experimental results have also illustrated the behavior of DNN partitioning under heterogeneous network conditions, highlighting the use-case of incrementally adding processing capacity. These results enable sizing of both DNN and network properties to achieve inference throughput improvements, prior to the deployment, with deterministic conditions on the necessary link throughputs, to enable a maximal inference throughput acceleration.

Chapter 7

Dynamic DNN partitioning

DNN partitioning has proven to enable inference throughput improvements on edge networks by pipe-lining the inference computation over available edge devices. Additionally, chapter 6 has identified bounds on both the attainable performance and complexity of the B&B based solver presented in chapter 5. Chapter 6 also provided guidelines on the influence of network properties on these values, in stable edge network environments.

However, in the context of edge computing, the underlying network is often subject to unreliable resources, unstable connectivity, and overall dynamicity [128]. While the B&B algorithm presented in chapter 5 yields optimal solutions for fixed DNN and network configurations, at a given point in time, this implies that a partitioning can become obsolete when changes occur. Such changes can be caused by network faults, as described in chapters 3 and 4 (*e.g.*, loss of connectivity, routing loops, blackouts, brownouts, packet drops), or concurrent access to the edge resources (*e.g.*, workload allocation on a node, or data transiting through the edge). This chapter extends the analysis to include behaviour over time, in order to understand the robustness and dependency to outside changes, *e.g.*, in computing and networking resources.

In this context, with the varying levels of complexity of B&B computations, it is important to understand the performance/cost trade-off of dynamic partitioning, and the influence of network parameters on this trade-off, in order to maintain a desired level of performance, while preserving the computing resources of the edge.

7.1 Related work

Dynamic DNN Surgery [98] considers two states of the underlying network, *i.e.*, a lightly and a heavily loaded condition, which trigger separate algorithms, respectively optimizing the inference delay and throughput. This is intended to lower the impact of DNN partitioning on the network.

PANDA [129], and CoEdge [130], are *adaptive* DNN partitioning solutions, *i.e.*, solutions which make partitioning decisions adaptively, gradually converging towards a solution which maximizes the defined optimization goal. Such methods can be extended during run-time to adapt to unstable network conditions, although they are not specifically built for this purpose.

The algorithm presented in [99] takes per-frame partitioning decisions, *i.e.*, two consecutive frames can be partitioned differently, *e.g.*, one frame can be computed

locally while another is offloaded to another device, or the cloud. This decision is also based on online appreciation of the state of the network, and can in turn be useful for dynamic partitioning. However, this category of work assumes independence of each frame, and would result in irregular frame rates, requiring further regularization mechanisms, outside of the scope of this study.

In sum, although some mechanisms have been explored to adapt to unstable network conditions, a reactive method, which provides stable DNN partitionings across frames, based on observability of the network, is still missing.

7.2 Statement of purpose

This chapter presents an evaluation of dynamic partitioning, *i.e.*, the re-evaluation of DNN partitioning over time, based on the methodology presented in chapter 5. The cost of a single B&B computation is compared with the potential gain in performance, in order to provide an insight into how to select ideal re-computation policies for different contexts. This chapter illustrates this trade-off through several dimensions: (i) the properties of the edge network, (ii) the desired reactivity and performance levels, (iii) the resources available for B&B computation, and (iv) the robustness to unstable network conditions.

In addition, this chapter details the principle elements of a dynamic DNN partitioning system design, as a guideline for a real-world implementation.

7.3 Chapter outline

This chapter is organised as follows: section 7.4 illustrates the motivations for dynamic DNN partitioning computations. Section 7.5 presents the dynamic partitioning system design, and compares re-computation policies to describe the trade-off between performance and computational cost.

7.4 Benefits of dynamic partitioning

This section presents the two benefits of dynamic partitioning: the improved prediction accuracy of B&B (section 7.4.1), and its increased resiliency to network perturbations (section 7.4.2).

7.4.1 Improving prediction accuracy

After deployment of an initial partitioning (cf. chapter 5), monitoring the effective transmission and inference latencies allows for their comparison with predicted values. When a re-computation is initiated, the estimated link throughput and processing rate values used by the B&B modeling are updated to fit the observed latencies.

Figure 7.1 illustrates this re-computation and its impact on the predicted and actual inference throughput values, in a network of $N = 2$ nodes[1] and a maximum number of split points $S = 3$. The blue points show the inverse values of measured

[1]Two NVIDIA Jetson Nanos connected via a standard Ethernet link.

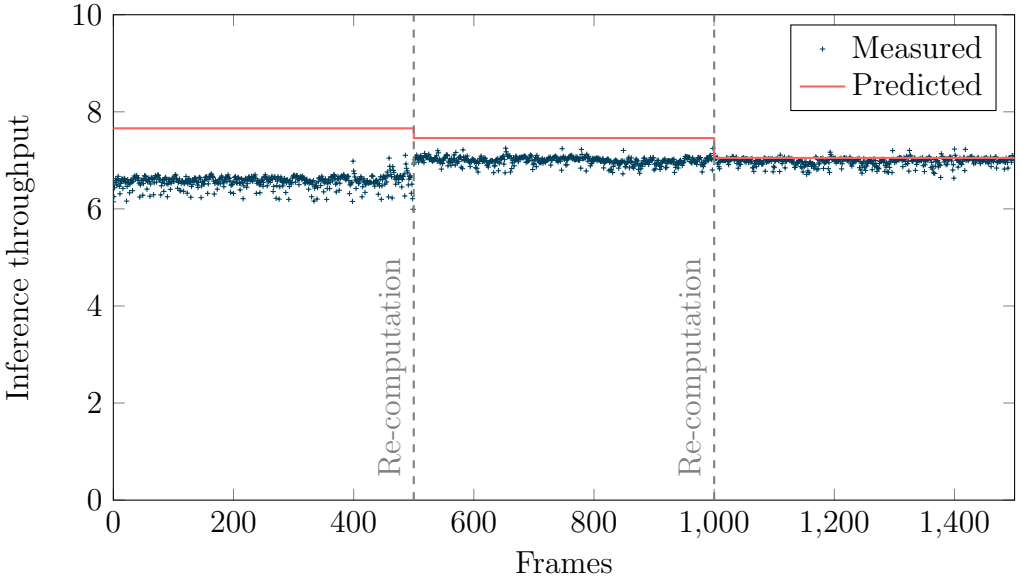


Figure 7.1: Refinement of the inference throughput precision through re-computations. At every re-computation, the processing rate and link throughput values are refined to better fit the observed latencies.

inference latencies, and the red line their average value between re-computations, *i.e.*, the measured inference throughput. The grey dotted vertical lines correspond to the re-computation times.

Figure 7.1 shows that, by triggering re-computations, and adjusting the link throughput and node processing rate values, (i) the precision of the predicted inference throughput, and (ii) the achieved inference throughput, both improve. This mechanism is an exploratory method to refine partitionings, as it gradually refines the network profiling of B&B, as partitions are displaced on the nodes.

7.4.2 Robustness to network variations

Dynamicity of edge computing environments can cause variations to occur on the effective node processing rates of the network. An example here is the allocation of an additional workload on one of the nodes performing the inference, and which will compete for access to the processing resource, and therefore will lower the effective processing rate seen by the DNN. The simulations presented in this section illustrate the behaviour of DNN partitioning when subject to such variations.

Processing rate variations

To isolate the contribution of this type of perturbation, the simulation will consider the following set-up: given a network of $N = 3$ nodes, and a maximum number of split points $S = 2$ in the B&B algorithm, the simulations will add, on a randomly selected node, a single new workload, occupying a ratio λ of the total processing resource of the node. This process is repeated for every node, and the resulting relative performance change of the DNN inference rate is averaged over all the simulations. Thus, the resulting performance drop will illustrate the average impact of a single workload allocation on the network running the distributed DNN. To illustrate the

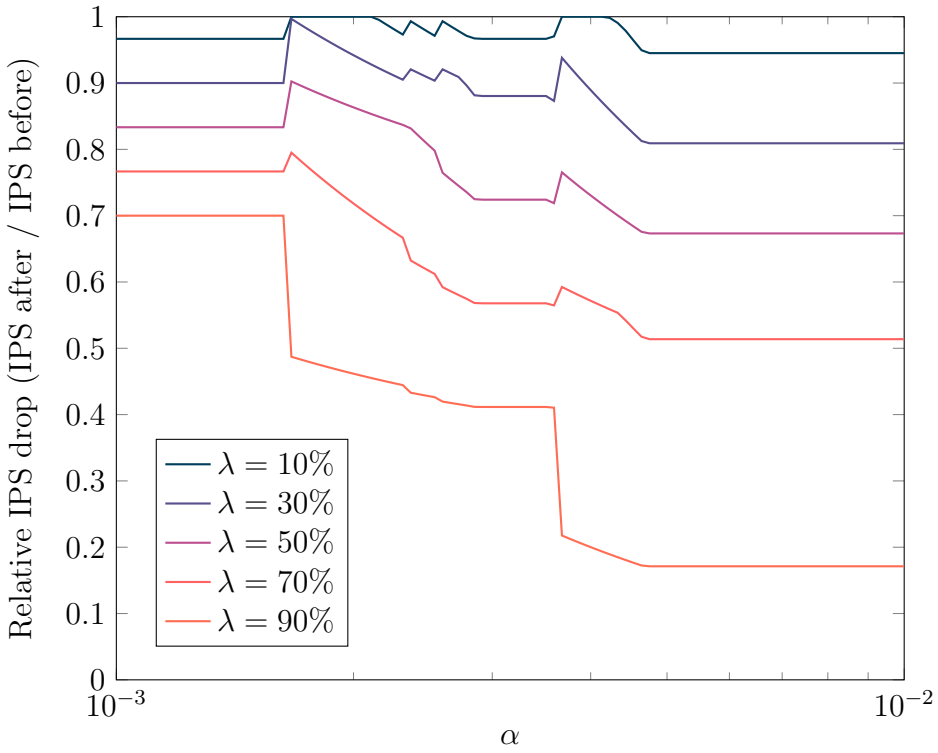


Figure 7.2: Inference throughput variations to processing rate variations, for a network of $N = 3$ nodes, a maximum number of splits $S = 2$ and varying node to processing rate ratios α.

influence of the network properties on the robustness to processing rate variations, the simulations are run using different values of the link throughput to processing rate ratio α, the behaviour being identical for different set-ups with identical α.

The metric used to measure the impact of these perturbations is the ratio between the inference throughput after insertion of the workload, and the initially observed inference throughput.

The simulation results are depicted in figure 7.2, which shows that the inference throughput performance drops, as expected, with increasing size of additional workloads. For values of α between the placement existence boundary and the optimal partitioning boundary (cf. section 6.3.2), and low processing rate variations, the inference throughput can remain unchanged, while it is systematically lowered outside of this region. This implies that partitioning a DNN can actually improve its robustness to network variations. Additionally, with values of α above the maximum complexity boundary, the inference throughput is systematically more sensitive to processing rate variations. This implies that during sizing edge networks for DNN partitioning, keeping a value of α between the two boundaries is preferable in robustness-sensitive use-cases, with the added benefit of keeping a low B&B complexity — section 6.3.2 having shown that complexity increases with α.

Nevertheless, for a majority of cases the performance is significantly impacted by processing rate variations, rendering the partitioning sub-optimal, and hereby motivating the need to re-compute the optimal partitioning.

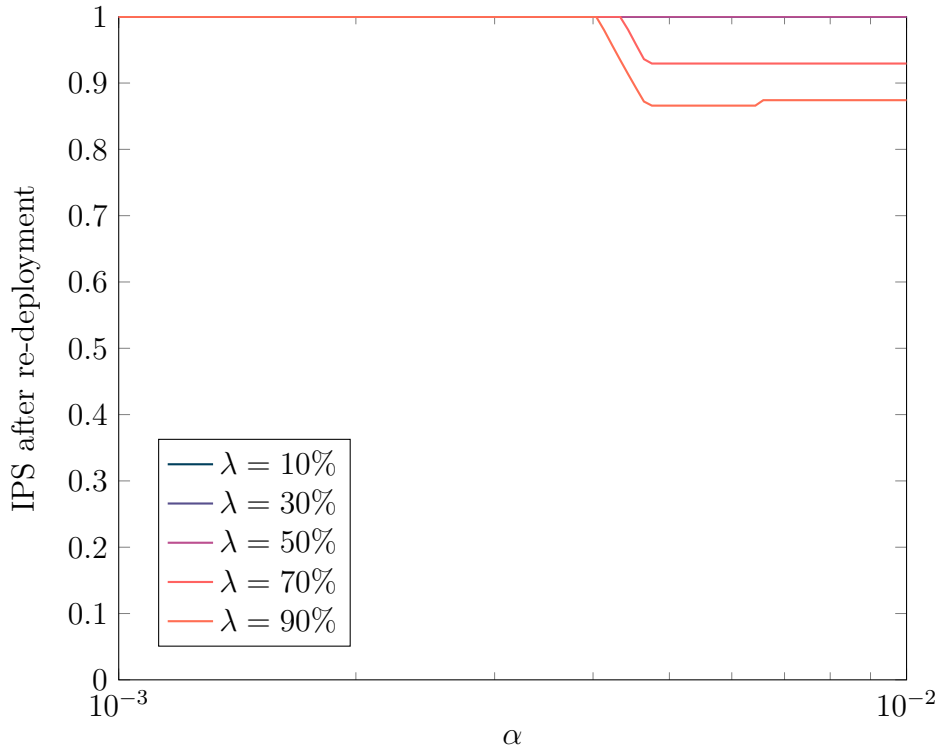


Figure 7.3: Achieved inference throughput after re-computation of the optimal DNN partitioning, with persistent perturbation. In this set-up, workloads need to occupy at least 70% of a node capacity before re-computation of the DNN partitioning cannot recover the pre-perturbation performance.

DNN partitioning re-computation

With the impact of network variations illustrated in the previous section, the following presents the attained inference throughput after re-computation of the optimal partitioning. The simulations consider the re-computation of B&B after insertion of a perturbation. The results are evaluated by the achieved inference throughput after re-computation, compared with its original value.

Figure 7.3 shows that with levels of variations below 70% of the node processing rate, re-deploying the DNN to the new optimal partitioning always recovers the initial inference throughput. This observation argues in favor of implementation of an adaptive partitioning strategy, because it maintains the optimal performance through perturbations.

7.5 Performance-cost trade-off

Sections 7.4.1 and 7.4.2 having illustrated the benefits of dynamic re-computation of the optimal partitioning, this section presents implementation details of the dynamic DNN partitioning system, and the resulting trade-off between performance and re-deployment cost

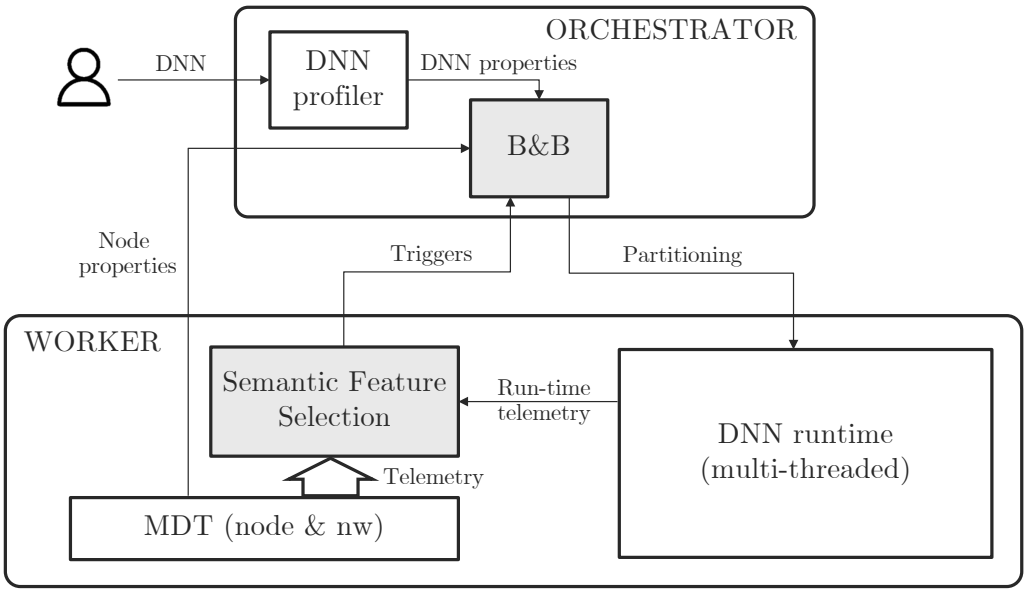


Figure 7.4: Functional diagram of the adaptive DNN partitioning system. The semantic feature selection tool is used to generate triggers for B&B re-computations whenever significant events occur on the network.

7.5.1 Dynamic partitioning system design

The dynamic DNN partitioning system components and implementation are presented in this section. The proposed adaptive partitioning system consists of two principal components:

- The *workers* run on every node, and contain the DNN run-time. Workers also contain a telemetry component that extracts information on the node, network, and the DNN run-time. Additionally, to limit the footprint of telemetry on the network bandwidth, workers contain a semantic feature selection component, which detects and extracts significant information from telemetry data to individually generate re-computation triggers for B&B.

- The *orchestrator* extracts the DNN information during the initialization step, and aggregates information about the nodes and links, from the workers. The orchestrator also implements the re-computation policy, *i.e.*, it can re-compute the DNN partitioning either periodically, based on run-time information, or when triggered by the workers.

This separation allows for the deployment of workers independently from the orchestrator, to form a managed cluster.

An example system diagram is shown in figure 7.4, where the re-computation policy is based on the fault diagnosis tools presented in part II. The semantic feature selection part of the worker in figure 7.4 extracts small selections of features summarizing network faults or events, and this selection is leveraged to generate triggers for B&B.

7.5.2 Re-computation policies

This section discusses re-computation policies which can be adopted to trigger re-evaluation of the optimal partitioning.

Periodic computation

This policy consists of triggering re-computation of the optimal DNN partitioning at regular time intervals. The benefit hereof is simplicity of implementation, and low impact on the network, since no active monitoring of the network characteristics is required, nor of the effective performance. However, periodic re-computations may lack reactivity when the chosen time interval is too long. Shortening this interval induces a higher computational footprint on the device running B&B. Thus, adopting a periodic policy creates a trade-off between reactivity and computational footprint, which depends on the re-computation interval.

Reactive re-computation

This policy consists of triggering re-computations when the performance differs significantly from the one expected by B&B. After initial computation of the best partitioning, the B&B algorithm has generated a partitioning expected to run at some predicted inference throughput value, and this re-computation policy triggers re-computations when the difference between achieved and predicted inference throughput values exceeds a certain threshold. Similarly to the previous method, this method has a low impact on the network because it does not involve active monitoring of its characteristics (*e.g.*, link throughputs and node processing rates), but offers varying levels of reactivity, depending on the threshold value.

Proactive re-computation

This policy consists of triggering re-computations based on a dedicated monitoring element. For example, this element can leverage the monitoring tools presented in part II, which extract significant events in telemetry data. The advantage of this method is that the triggers will only be generated when significant and persistent changes occur on the network. Depending on implementation, implies a significant impact on the computational resources of the network, because it requires active extraction and monitoring of telemetry data. For example, this event-triggered method can rely on an online change-point detection method [87] on telemetry data extracted from the nodes.

7.5.3 Re-computation cost

To evaluate the trade-off between the performance and cost of dynamic partitioning, this section describes (i) the implied latencies and performance cost, and (ii) the computational impact of re-computation.

Re-computation latency

The different phases of adaptive DNN partitioning are shown in figure 7.5, in the context of the system reacting to a perturbation. This section narrows its

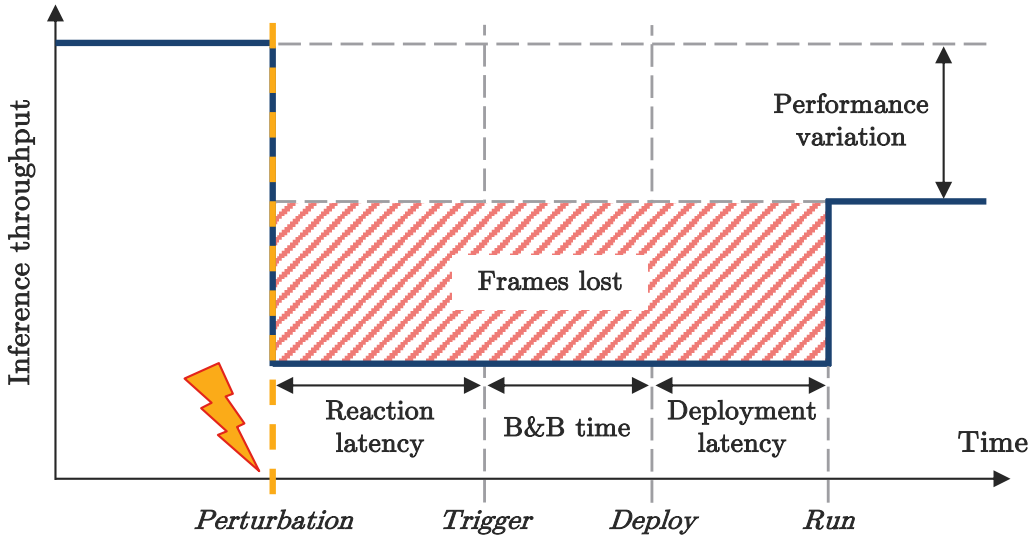


Figure 7.5: Metrics for adaptive DNN partitioning.

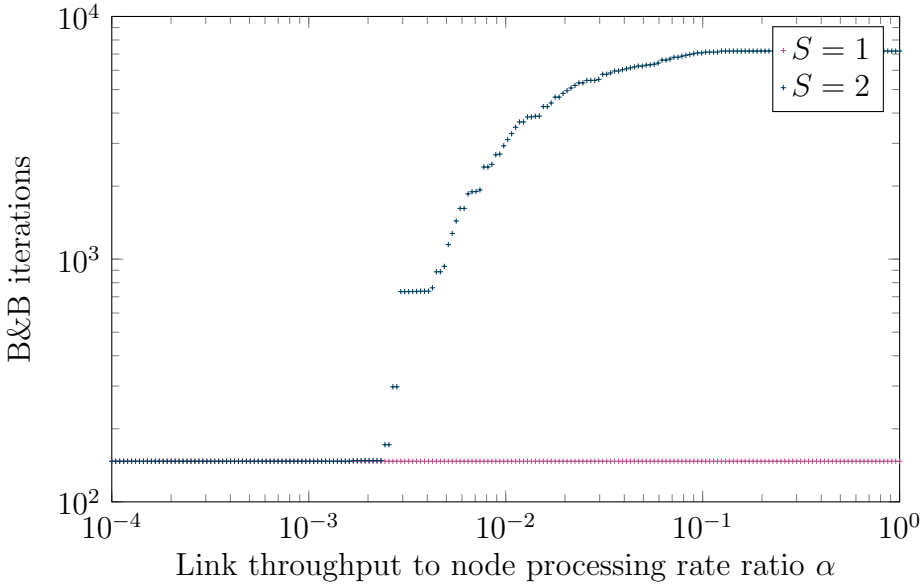
analysis on two principal latencies: (i) the reaction latency, *i.e.*, the time between the start of the perturbation and the B&B trigger, and (ii) the B&B computation time. This model ignores the re-deployment latency, *i.e.*, the time required to stop computation, move layers between nodes, and re-start the inference, because of its strong dependency on implementation details.

Therefore, to evaluate and compare re-computation policies, the simulations will use the following metrics:

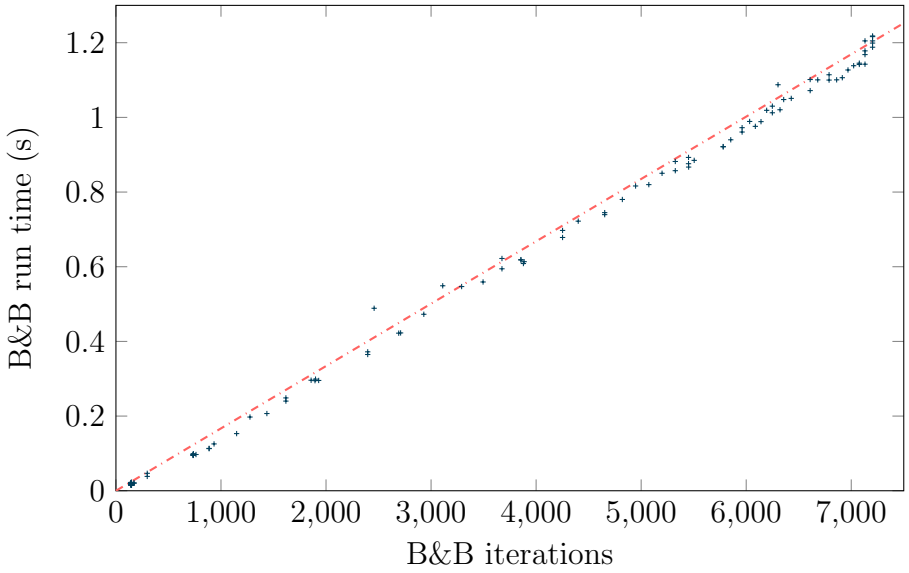
- **The reaction latency**, *i.e.*, the time between the perturbation and the trigger of B&B.
- **The performance variation**, represented in figure 7.3, *i.e.*, the ratio between the inference throughput after re-deployment, and the initial inference throughput.
- **The number of lost frames**, *i.e.*, the number of frames lost compared to the optimal partitioning, represented by the dashed area in figure 7.5

Computational cost

With the assumption that B&B is used as the re-computation algorithm, this section presents an estimation of the cost to recompute a single DNN partitioning. As shown in figure 6.2, the number of partitionings evaluated by B&B, which is linked to its complexity, only depends on the link throughput to node processing rate ratio α. This relationship is depicted in figure 7.6(a) for a YOLOv2 DNN model, a network of 3 nodes, and varying maximum allowed numbers of split points S. This illustrates the possibility to estimate B&B iterations based solely on properties of the network. Additionally, with figure 7.6(b) illustrating the linear relationship between B&B iterations and the corresponding CPU cycles required to run the computation, it is possible to estimate the cost of this computation and its implied delay, which are the important metrics to estimate the cost of a re-computation.



(a) Influence of α on B&B iterations



(b) Influence of B&B iterations on CPU cycles

Figure 7.6: B&B cost to compute estimation. Figure 7.6(a) shows the correspondence between B&B iterations and properties of the network ,through the link throughput to node processing rate ratio α. Figure 7.6(b) shows le linear relationship between B&B iterations and their required CPU cycles.

7.5.4 Comparison and discussion

With the re-computation cost described in sections 7.5.2 and 7.5.3, dimensioning an adaptive partitioning system consists of identifying the trade-off between computational footprint and reactivity to faults or events persistently changing the state of the network. A comparison of the different re-computation policies is summarize in table 7.1.

In addition to this comparison, it is important to note the importance of the link throughput to node processing rate ratio α in the performance-cost trade-off of dynamic partitioning. This ratio determines the cost of a single B&B iteration, as shown in figure 7.6, which sets the limiting computational factor between B&B computation, and telemetry monitoring. Namely, in networks with low values of α, the cost to run a single B&B computation is low, which argues in favor of periodic re-computations, enabling better B&B prediction accuracy, and potentially improved performance. Alternatively, in networks with a high α value, the cost of a single B&B computation is too important, which argues in favor of a reactive or proactive re-computation policy, depending on the available resources for telemetry monitoring.

7.6 Summary of results

This chapter has identified the caveat of designing dynamic DNN partitioning systems. This chapter has shown that a dynamic DNN Partitioning scheme can not only allow to recover from network perturbations impacting the final DNN inference throughput, but can also compensate for the inference and transmission latency prediction inaccuracies of the B&B modeling, enabling an increased precision and achieved inference throughput. With both of these advantages highlighted, this chapter has then described an example dynamic partitioning system design, with periodic, reactive, and proactive re-computation policy options. These re-computation policies were finally evaluated on the identified cost and performance metrics.

This comparative analysis has allowed to identify the essential factor in the dimensioning of a dynamic partitioning system: the link throughput to node processing rate ratio. Once again, this ratio being linked to the B&B complexity, it determines the relative cost of B&B computations compared with the cost of active monitoring of network properties, to trigger re-computations. These factors argue in favor of frequent passive re-computations in low α networks, and reactive or proactive re-computations, in high α networks, where the cost of B&B is high.

In sum, part III has demonstrated the importance of the link throughput to node processing rate ratio in DNN partitioning at the edge. In both static and dynamic cases, it is critical for operators to tune this ratio on their edge networks, to fit their desired performance, complexity, and adaptability to network variations.

Re-computation policy	**Periodic**	**Reactive**	**Proactive**
B&B footprint	**Very high**: B&B computation every T seconds.	**Low**: B&B computations only occur with significant performance drops.	**Low**: B&B computations only occur with significant network events.
Telemetry footprint	**None**	**Low**: active monitoring on a single metric (inference throughput).	**Very high**: Every node actively monitors its telemetry data to detect events.
Prediction precision	**Very high**: Frequent re-computation of the optimal partitioning regularly refines the inference throughput prediction.	**Low**: The triggers only occur in reaction to network events, not for refinement.	**Low**: The triggers only occur in reaction to network events, not for refinement.
Reaction latency	**High**: The reaction latency is on average of $T/2$, T being the re-computation period.	**Low**: The reaction latency depends on the chosen threshold value and speed of performance degradation.	**Very low**: Proactively monitoring telemetry enables fast reaction times.
Detection accuracy	Very high number of false positives, *i.e.*, unnecessary re-computations.	High accuracy. Some false positives can occur when non-persistent perturbations occur on the network.	High accuracy. Potential false positives in cases where significant network changes do not impact performance.

Table 7.1: Comparison of re-computation policies for dynamic DNN partitioning.

Part IV

Conclusion

Chapter 8

Conclusion

Smart devices have become increasingly common due to the widespread availability and affordability of computing and networking technologies. These devices, ranging from smart home appliances to wearable technology, offer convenience and efficiency in daily tasks. They hold the potential to enhance safety and security through real-time monitoring and alerts. This growing adoption of smart devices has led to the generation of increasing amounts of data. In this context, the evolution of distributed computing has been driven by the need to handle this vast data influx, and led a wide adoption of cloud computing in the 20210s, offering large controllable resources on-demand, to support heavy workload computation. However, external factors, such as privacy requirements and low-latency demands are incompatible with cloud computing deployments. For example, self-driving cars rely heavily on Artificial Intelligence but must process data close to the source due to latency constraints and privacy regulations. To address these limitations, edge computing, a computing paradigm in which computation is performed in close proximity to data source, has emerged. Edge computing relies on the computing capacity of smart devices, close to data sources, to provide services which maintain the convenience and performance of the cloud. The central question investigated in this thesis is how to deploy and monitor resource-intensive applications in resource constrained edge networks, while maintaining the user experience associated with cloud computing.

This thesis has shown that the multiplicity and heterogeneity of edge devices can be leveraged to enable accelerations at the smart device edge. This approach has shown to provide (i) observability on network events occurring in high dimensional heterogeneous telemetry data, and (ii) throughput acceleration for heavy pipe-lined workloads. The benefits of this approach have been demonstrated through theoretical results, simulations, and evaluation of experimental results in realistic environments.

The problem of observability on edge networks has been investigated through the lens of event and fault diagnosis on network telemetry data. The methods and metrics presented in this section facilitate the diagnosis of faults and events on an edge network, by providing short and semantically meaningful hints to network operators. Chapter 3 studied the question of information contained in edge device telemetry data, through the example of network streaming telemetry. With the problem of volume identified in MDT data, this chapter has defined a metric to estimate the importance relationships in observability data, in order to filter the relevant information out of large volumes of data. In chapter 4, the information

theory based approach described in chapter 3 is leveraged to define a semantic feature selection method to help diagnosing events in network telemetry. After having identified the lack of semantic understanding of data-driven methods for fault diagnosis, the efficiency and robustness of this method is presented as a way to create intelligible selections of features, to explain events occurring on an edge network. This work on observability in network telemetry data has led to the publication of two conference papers [52, 53].

With these observability mechanisms to understand the state of the network, this thesis has further studied methods for performance improvement at the edge, through the lens of throughput acceleration of heavy pipe-lined workloads at the smart device edge, with the example of DNN partitioning. Chapter 5 has presented (i) working assumptions and modeling to predict inference and transmission latencies, inference throughput, and (ii) the B&B algorithm to find partitioning which maximize the inference throughput. In chapter 6, simulations and experiments have allowed for the definition of deterministic bounds on the performance and complexity of the DNN partitioning problem. These conditions define the scope of applicability of this method and allows developers to size their edge network to effectively achieve the best compromise between performance and complexity. Because of the overall dynamicity of edge network properties, chapter 7 presents an analysis of the robustness and adaptability of DNN partitioning, describing a dynamic partitioning system design and re-computation policies. Once again, the properties of the edge network are deterministic in the system design and the insights provided in this chapter enable the optimization of performance, robustness, complexity, and adaptability of a dynamic partitioning system. This work has led to the publication of a journal paper [54].

In sum, the work provided in this thesis demonstrated the possibility to provide mechanisms leveraging the available computational capacity of the smart device edge to enhance the capacity of the edge. Future work in this field should focus on several key aspects to further advance the deployment and monitoring of resource-intensive applications in resource-constrained edge networks while maintaining a cloud-like user experience. Firstly, investigations could be carried out to explore the potential of hybrid edge-cloud architectures. By integrating the strengths of both edge computing and cloud computing, such architectures may offer a more flexible and scalable solution to meet the varying demands of different applications. Secondly, the development of more efficient and intelligent edge resource management techniques is crucial. This includes exploring advanced machine learning algorithms to optimize resource allocation and workload distribution dynamically based on real-time conditions and requirements, in place or in addition to the semantic feature selection method for event and fault diagnosis. Furthermore, the security and privacy challenges associated with edge computing need to be addressed comprehensively. As the edge deals with sensitive data, ensuring data privacy and protection from potential security threats is paramount. Future work could involve the development of robust encryption and authentication mechanisms tailored for edge environments. Lastly, as edge computing technologies evolve, it is crucial to conduct extensive real-world deployments and large-scale testing to validate the effectiveness and practicality of proposed solutions. Case studies and field experiments in diverse environments would provide valuable insights into the performance, scalability, and reliability of edge computing systems.

Overall, addressing these challenges and exploring the opportunities for synergy between edge and cloud computing will pave the way for a more robust and efficient edge ecosystem. This will enable the realization of the full potential of smart devices and edge computing, leading to a paradigm shift in the way we interact with and benefit from the vast amount of data generated by the ever-expanding array of smart devices.

Appendix A

Résumé en français

Cette thèse étudie l'accélération de charges de travail lourdes dans les réseaux d'appareils intelligents en périphérie de réseau, en se concentrant sur les problématiques (i) d'observabilité et de filtrage de données de télémétrie, ainsi que (ii) une méthodologie de pipelining pour accélérer le débit de traitement d'importantes charges de travail. Elle comprend 4 parties et 8 chapitres, structurés comme suit.

La partie I fournit une discussion introductive. Le chapitre 1 présente les évolutions en conception d'applications et en informatique distribuée conduisant à l'émergence de l'informatique en périphérie, ainsi que les contraintes et limitations de ce genre environnement. Puis, deux méthodologies permettant d'améliorer les performances d'appareils intelligents sont présentées, dans le contexte de l'informatique en périphérie: les approches cloud-out et edge-in, qui examinent respectivement la relocalisation de charges de travail et l'amélioration des capacités d'appareils intelligents, afin d'obtenir de meilleures performances. Enfin, une discussion est menée sur la manière dont l'augmentation des capacités des appareils intelligents plaide en faveur d'une approche hybride, qui exploite la multiplicité des appareils en périphérie de réseau pour localement distribuer la charge de travail.

La partie II étudie l'observabilité de l'état du réseau et le diagnostic de pannes en périphérie de réseau. Les systèmes experts requièrent d'importants efforts de construction et de maintenance, et manquent à la fois de scalabilité et d'adaptabilité aux événements inconnus ou aux modifications de la topologie du réseau. Dans ce contexte, le chapitre 3 (publié dans [52]) présente une méthode basée sur les données pour extraire des sélections intelligibles de caractéristiques opérationnelles à partir de données de télémétrie de réseau de haute dimension, afin de fournir une visibilité sur l'état du réseau, et de faciliter le diagnostic de défaillance pour les opérateurs. La méthode présentée est basée sur l'extraction d'informations à partir des métadonnées contenues dans les noms de caractéristiques, à travers l'exemple de la télémétrie pilotée par modèle (MDT) et le langage de modélisation YANG. Ceci est réalisé en quantifiant les informations préservées dans la sélection de caractéristiques danas le contexte du diagnostic, grâce à une mesure d'entropie croisée, définie sur un espace de mots contenus dans la nomenclature YANG. Cette mesure conduit à la définition d'un score de qualité de sélection, en faveur de sélections préservant des informations intelligibles pour les opérateurs de réseau.

Avec cette mesure d'importance sémantique définie, le chapitre 4 présente une

évaluation de la sélection sémantique de fonctionnalités dans le contexte du diagnostic de panne (publié dans [53]). Ce chapitre préconise l'utilisation d'une analyse sémantique dans le diagnostic, après avoir illustré les lacunes de trois catégories de méthodes basées uniquement sur des données de télémétrie. Ces mécanismes manquent en compréhension d'une importance sémantique dans les données et bénéficieraient de cette analyse sémantique supplémentaires. En utilisant la méthodologie du chapitre 3, ce chapitre explore l'hypothèse selon laquelle une partie de ces connaissances supplémentaires peut être extraite des métadonnées. L'approche proposée combine des mesures basées sur les données et des informations sémantiques contenues dans les noms de fonctionnalités pour produire des sélections qui représentent au mieux un événement sous-jacent. Cette étude étend l'estimation d'importance sémantique basée sur l'entropie croisée à la définition d'un problème d'optimisation qui associe cette importance sémantique au comportement des données. Une architecture de référence est ensuite introduite pour évaluer les avantages de cette analyse sémantique et démontrer la performance et la robustesse de la sélection sémantique de fonctionnalités, sur différents types de pannes, dans des ensembles de données de télémétrie réseau, modélisés avec le langage de modélisation de données YANG. Les résultats illustrent l'intérêt d'une telle analyse complémentaire pour le diagnostic et mettent en évidence la robustesse de l'approche étudiée face aux variations de données et métadonnées d'entrée. L'addition d'une étude sémantique dans un processus de diagnostic permet des avantages exceptionnels en termes d'exhaustivité des sélections, ainsi qu'une suppression importante de sélections parasite (*i.e.*, une augmentation de la précision).

La partie III étudie la **répartition d'importantes charges de travail** dans l'informatique de périphérie, à travers l'exemple de l'accélération de débit d'inférence de réseaux de neurones profonds (DNN) (publié dans [54]). L'inférence sur des données en temps réel nécessite d'importantes ressources pour satisfaire les exigences des applications. Cependant, les applications d'apprentissage profond sensibles à la latence et à la confidentialité, *e.g.*, dans les cas d'utilisation de l'informatique de périphérie, ne peuvent pas se permettre de décharger le calcul vers des clouds distants en raison de l'important coût de transmission et du manque de confiance envers les fournisseurs de cloud tiers. Cependant, dans les mécanismes d'accélération standards, l'accélération matérielle peut être onéreuse et l'optimisation du modèle nécessite des efforts de conception importants, tout en entravant la précision du modèle. Dans le chapitre 5, une méthode de partitionnement de réseaux neuronaux profonds est présentée comme une troisième approche complémentaire, qui consiste à répartir la charge de travail d'inférence sur plusieurs appareils intelligents disponibles, en tenant compte des propriétés du réseau et de la structure du réseaux neuronal, dans le but de maximiser le débit d'inférence (nombre d'inférences par seconde). Ce chapitre introduit une méthode pour prédire les latences d'inférence et de transmission dans des déploiements de réseaux de neurones distribués multithreadés et met en évidence les problèmes liés à l'exactitude de ces prédiction, pour cause d'optimisation du modèle et d'accélération matérielle hétérogènes dans l'environnement d'exécution. Avec cette représentation du comportement d'inférence, ce chapitre décrit explicitement les potentiels objectifs de niveau de service pour le partitionnement de DNN, *i.e.*, la latence de bout en bout, la consommation d'énergie et le débit d'inférence. Après avoir identifié que ce problème était un

problème d'optimisation linéaire en nombres entiers (OLNE), ce chapitre décrit un processus d'optimisation basé sur une méthode de séparation et évaluation (S&E) dans le but d'identifier des partitions qui maximisent le débit d'inférence du réseau de neurone, tout en gardant le calcul sur le réseau de périphérie. Cette méthode est comparée à des solutions de problèmes d'OLNE standards pour prouver que S&E convient mieux au contexte de l'informatique de périphérie, car il résout rapidement les cas simples. Un mécanisme d'arrêt précoce est décrit afin de limiter sa complexité dans les cas extrêmes.

Dans le chapitre 6, le solveur de partitionnement S&E est analysé pour quantifier les limites de sa performance et de sa complexité. Tout d'abord, des simulations sont présentées dans un environnement de réseau homogène (nœuds et liens identiques) pour explorer l'influence des paramètres d'entrée du problème sur le débit d'inférence atteint et la complexité de S&E. Les premières simulations mettent en évidence l'influence du nombre de nœuds et du nombre maximal de partitions sur les performances et la complexité, ce qui conduit à l'identification d'une limite sur le nombre de points de division nécessaires, permettant ainsi aux opérateurs de limiter la complexité de S&E. Les deuxièmes simulations évaluent l'influence des nœuds et liens sur le réseau, montrant que le nombre effectif d'itérations de S&E dépend d'un seul paramètre, à savoir le ratio entre débit des liens et vitesse de traitement des nœuds. De plus, ces simulations identifient trois valeurs de ce paramètre qui délimitent des régions aux comportements différents en termes de performance et de complexité. Cette analyse conduit à la définition de la *région d'accélération*, qui décrit les conditions déterministes sur les propriétés du réseau de neurone et du réseau dans lesquelles le partitionnement est bénéfique. Enfin, des résultats expérimentaux sont présentés pour confirmer les simulations et montrer des améliorations de débit d'inférence dans des exemples de déploiements. Les résultats montrent également une configuration hétérogène dans laquelle la performance approche la borne supérieure théorique précédemment définie. Le chapitre se termine par un résumé des résultats, qui peuvent être utilisés par les opérateurs et les développeurs d'applications pour dimensionner leur réseau et profiter au mieux du partitionnement de réseaux de neurones, avant le déploiement.

Au chapitre 7, un système de **partitionnement dynamique de réseau de neurone** est présenté, afin d'étudier la robustesse du partitionnement et son comportement dans des environnements réseaux instables. Étant donné l'importante dynamicité des réseaux de périphérie, ce chapitre illustre l'importance de réévaluations régulières du partitionnement, étant donné que la méthodologie des chapitres précédents ne s'appliquent qu'à des conditions de réseau stables. Tout d'abord, ce chapitre illustre les deux principaux avantages de la réévaluation dynamique: (i) la capacité de compenser l'imprécision dans la prédiction de latence d'inférence, en apprenant de valeurs mesurées pendant l'exécution, et (ii) la capacité de s'adapter à différents niveaux de perturbations sur le réseau. Dans ce contexte, les résultats de simulation présentés montrent que le partitionnement permet une meilleure robustesse aux perturbations, et que le partitionnement dynamique permet à la performance de rester optimale, même pour des perturbations importantes. Ensuite, ce chapitre fournit des indications sur la conception d'un système de partitionnement dynamique et des politiques de réévaluation. Trois politiques sont présentées et comparées: (i) une politique de réévaluation périodique, (ii) une politique de réévaluation réactive, qui réévalue le partitionnement lorsque le débit d'inférence tombent en dessous d'un

seuil, et (iii) une politique de réévaluation proactive, qui repose sur la surveillance des données de télémétrie pour déclencher des réévaluations. Ce chapitre conclut en identifiant le même ratio entre débit des liens et taux de traitement des nœuds comme un facteur clé dans le choix de la politique de réévaluation, car il définit le ratio entre l'empreinte computationnelle de S&E et celle de la surveillance active de la télémétrie de réseau.

Enfin, la partie IV conclut ce manuscrit.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] S. Mochizuki, K. Matsubara, K. Matsumoto, C. L. P. NGuyen, T. Shibayama, K. Iwata, K. Mizumoto, T. Irita, H. Hara, and T. Hattori, "A 197mw 70ms-latency full-hd 12-channel video-processing soc in 16nm cmos for in-vehicle information systems," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E100.A, no. 12, pp. 2878–2887, 2017.

[2] European Commission, "Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance)," 2016.

[3] F. Douglis and M. Kaashoek, "Scalable internet services," *IEEE Internet Computing*, vol. 5, no. 4, pp. 36–37, 2001.

[4] A. Vakali and G. Pallis, "Content delivery networks: status and trends," *IEEE Internet Computing*, vol. 7, no. 6, pp. 68–74, 2003.

[5] J. Joseph, M. Ernest, and C. Fellenstein, "Evolution of grid computing architecture and grid adoption models," *IBM Systems journal*, vol. 43, no. 4, pp. 624–645, 2004.

[6] R. Schollmeier, "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications," in *Proceedings First International Conference on Peer-to-Peer Computing*, pp. 101–102, IEEE, 2001.

[7] P. Mell, T. Grance, *et al.*, "The nist definition of cloud computing," *NIST Special Publication 800-145*, 2011.

[8] L. Edge, "White paper: Sharpening the edge: Overview of the lf edge taxonomy and framework," tech. rep., LF Edge, 2020.

[9] A. D. Kshemkalyani and M. Singhal, *Distributed computing: principles, algorithms, and systems*. Cambridge University Press, 2011.

[10] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing.," in *NSDI* (S. D. Gribble and D. Katabi, eds.), pp. 15–28, USENIX Association, 2012.

[11] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[12] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[13] H. Lin, S. Zeadally, Z. Chen, H. Labiod, and L. Wang, "A survey on computation offloading modeling for edge computing," *Journal of Network and Computer Applications*, vol. 169, no. July, p. 102781, 2020.

[14] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[15] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards wearable cognitive assistance," in *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pp. 68–81, 2014.

[16] J. Zhu, D. S. Chan, M. S. Prabhu, P. Natarajan, H. Hu, and F. Bonomi, "Improving web sites performance using edge servers in fog computing architecture," in *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*, pp. 320–323, 2013.

[17] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Transactions on Wireless Communications*, vol. 16, no. 3, pp. 1397–1411, 2017.

[18] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Transactions on Communications*, vol. 64, no. 10, pp. 4268–4282, 2016.

[19] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[20] Y. Liu, D. Niu, and B. Li, "Delay-optimized video traffic routing in software-defined interdatacenter networks," *IEEE Transactions on Multimedia*, vol. 18, no. 5, pp. 865–878, 2016.

[21] F. Guo, H. Zhang, H. Ji, X. Li, and V. C. M. Leung, "An efficient computation offloading management scheme in the densely deployed small cell networks with mobile edge computing," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2651–2664, 2018.

[22] S. Bi and Y.-J. A. Zhang, "An admm based method for computation rate maximization in wireless powered mobile-edge computing networks," in *2018 IEEE International Conference on Communications (ICC)*, pp. 1–7, 2018.

[23] S. Bi and Y. J. Zhang, "Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading," *IEEE Transactions on Wireless Communications*, vol. 17, no. 6, pp. 4177–4190, 2018.

[24] Y. Xiao and M. Krunz, "Qoe and power efficiency tradeoff for fog computing networks with fog node cooperation," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pp. 1–9, 2017.

[25] L. Liu, Z. Chang, X. Guo, S. Mao, and T. Ristaniemi, "Multiobjective optimization for computation offloading in fog computing," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 283–294, 2018.

[26] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. S. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Transactions on Communications*, vol. 65, no. 8, pp. 3571–3584, 2017.

[27] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.

[28] G. Zhang, W. Zhang, Y. Cao, D. Li, and L. Wang, "Energy-delay tradeoff for dynamic offloading in mobile-edge computing system with energy harvesting devices," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4642–4655, 2018.

[29] G. Zhang, Y. Chen, Z. Shen, and L. Wang, "Energy management for multi-user mobile-edge computing systems with energy harvesting devices and qos constraints," in *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–6, 2018.

[30] X. Lyu, W. Ni, H. Tian, R. P. Liu, X. Wang, G. B. Giannakis, and A. Paulraj, "Optimal schedule of mobile edge computing for internet of things using partial information," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2606–2615, 2017.

[31] W. Chen, D. Wang, and K. Li, "Multi-user multi-task computation offloading in green mobile edge cloud computing," *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 726–738, 2019.

[32] H. Ko, J. Lee, and S. Pack, "Spatial and temporal computation offloading decision algorithm in edge cloud-enabled heterogeneous networks," *IEEE Access*, vol. 6, pp. 18920–18932, 2018.

[33] M. Kamoun, W. Labidi, and M. Sarkiss, "Joint resource allocation and offloading strategies in cloud enabled cellular networks," in *2015 IEEE International Conference on Communications (ICC)*, pp. 5529–5534, 2015.

[34] W. Labidi, M. Sarkiss, and M. Kamoun, "Energy-optimal resource scheduling and computation offloading in small cell networks," in *2015 22nd International Conference on Telecommunications (ICT)*, pp. 313–318, 2015.

[35] Z. Wei, B. Zhao, J. Su, and X. Lu, "Dynamic edge computation offloading for internet of things with energy harvesting: A learning method," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4436–4447, 2019.

[36] J. Xu, L. Chen, and S. Ren, "Online learning for offloading and autoscaling in energy harvesting mobile edge computing," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 3, pp. 361–373, 2017.

[37] X. Zheng, M. Li, M. Tahir, Y. Chen, and M. Alam, "Stochastic computation offloading and scheduling based on mobile edge computing," *IEEE Access*, vol. 7, pp. 72247–72256, 2019.

[38] D. Van Le and C.-K. Tham, "An optimization-based approach to offloading in ad-hoc mobile clouds," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pp. 1–6, 2017.

[39] C. J. C. H. Watkins, *Learning from delayed rewards*. PhD thesis, King's College, Cambridge United Kingdom, 1989.

[40] L. T. Tan and R. Q. Hu, "Mobility-aware edge caching and computing in vehicle networks: A deep reinforcement learning," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 11, pp. 10190–10203, 2018.

[41] N. Cheng, F. Lyu, W. Quan, C. Zhou, H. He, W. Shi, and X. Shen, "Space/aerial-assisted computing offloading for iot applications: A learning-based approach," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1117–1129, 2019.

[42] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2016.

[43] H. Cao and J. Cai, "Distributed multiuser computation offloading for cloudlet-based mobile cloud computing: A game-theoretic machine learning approach," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 1, pp. 752–764, 2018.

[44] L. Liu, Z. Chang, and X. Guo, "Socially aware dynamic computation offloading scheme for fog computing system with energy harvesting devices," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1869–1879, 2018.

[45] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker, "Agile application-aware adaptation for mobility," *ACM SIGOPS Operating Systems Review*, vol. 31, no. 5, pp. 276–287, 1997.

[46] M. Weiser, "The computer for the 21st century," *ACM SIGMOBILE mobile computing and communications review*, vol. 3, no. 3, pp. 3–11, 1999.

[47] M. Satyanarayanan, "Pervasive computing: Vision and challenges," *IEEE Personal communications*, vol. 8, no. 4, pp. 10–17, 2001.

[48] M. R. Ebling, "Pervasive computing and the internet of things," *IEEE Pervasive Computing*, vol. 15, no. 1, pp. 2–4, 2016.

[49] OpenFog Consortium and Architecture Working, "OpenFog reference architecture for fog computing," tech. rep., OpenFog, February 2017.

[50] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13–16, 2012.

[51] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," *Proceedings of the International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, vol. 2015-June, pp. 37–42, 2015.

[52] T. Feltin, P. Foroughi, W. Shao, F. Brockners, and T. H. Clausen, "Semantic feature selection for network telemetry event description," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–6, IEEE, 2020.

[53] T. Feltin, J. A. Cordero Fuertes, F. Brockners, and T. H. Clausen, "Understanding semantics in feature selection for fault diagnosis in network telemetry data," in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, IEEE, 2023.

[54] T. Feltin, L. Marché, J.-A. Cordero-Fuertes, F. Brockners, and T. H. Clausen, "Dnn partitioning for inference throughput acceleration at the edge," *IEEE Access*, pp. 1–1, 2023.

[55] A. Morton, "Active and Passive Metrics and Methods (with Hybrid Types In-Between)." RFC 7799, May 2016.

[56] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, Z.-W. Lin, and V. Kurien, "Pingmesh: A large-scale system for data center network latency measurement and analysis," *SIGCOMM Comput. Commun. Rev.*, vol. 45, p. 139–152, aug 2015.

[57] K. Agarwal, E. Rozner, C. Dixon, and J. Carter, "Sdn traceroute: Tracing sdn forwarding without changing network behavior," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, (New York, NY, USA), p. 145–150, Association for Computing Machinery, 2014.

[58] B. Claise, "Cisco Systems NetFlow Services Export Version 9." RFC 3954, Oct. 2004.

[59] M. Wang, B. Li, and Z. Li, "sflow: Towards resource-efficient and agile service federation in service overlay networks," in *24th International Conference on Distributed Computing Systems, 2004. Proceedings.*, pp. 628–635, IEEE, 2004.

[60] P. Aitken, B. Claise, and B. Trammell, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information." RFC 7011, Sept. 2013.

[61] M. Fedor, M. L. Schoffstall, J. R. Davin, and D. J. D. Case, "Simple Network Management Protocol (SNMP)." RFC 1157, May 1990.

[62] R. Gerhards, "The Syslog Protocol." RFC 5424, Mar. 2009.

[63] A. Bierman and B. Lengyel, "With-defaults Capability for NETCONF." RFC 6243, June 2011.

[64] E. M. Bjorklund, "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)," RFC 6020, RFC Editor, October 2010.

[65] S. Solorio-Fernández, J. A. Carrasco-Ochoa, and J. F. Martínez-Trinidad, "A review of unsupervised feature selection methods," *Artificial Intelligence Review*, 2019.

[66] S. Alelyani, H. Liu, and L. Wang, "The effect of the characteristics of the dataset on the selection stability," in *2011 IEEE 23rd International Conference on Tools with Artificial Intelligence*, pp. 970–977, Nov 2011.

[67] J. Yao, Q. Mao, S. Goodison, V. Mai, and Y. Sun, "Feature selection for unsupervised learning through local learning," *Pattern Recognition Letters*, vol. 53, pp. 100 – 107, 2015.

[68] M. Dash, H. Liu, and J. Yao, "Dimensionality reduction of unsupervised data," in *Proceedings Ninth IEEE International Conference on Tools with Artificial Intelligence*, pp. 532–539, Nov 1997.

[69] R. Varshavsky, A. Gottlieb, M. Linial, and D. Horn, "Novel Unsupervised Feature Filtering of Biological Data," *Bioinformatics*, vol. 22, pp. e507–e513, 07 2006.

[70] V. M. Rao and V. N. Sastry, "Unsupervised feature ranking based on representation entropy," in *2012 1st International Conference on Recent Advances in Information Technology (RAIT)*, pp. 421–425, March 2012.

[71] P. Mitra, C. A. Murthy, and S. K. Pal, "Unsupervised feature selection using feature similarity," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, pp. 301–312, March 2002.

[72] A. Aizawa, "An information-theoretic perspective of tf-idf measures," *Information Processing and Management*, vol. 39, no. 1, pp. 45–65, 2003.

[73] D. Katz and D. Ward, "Bidirectional Forwarding Detection (BFD) for IPv4 and IPv6 (Single Hop)." RFC 5881, June 2010.

[74] "Telemetry Configuration Guide for Cisco NCS 5500 Series Routers, IOS XR Release 6.1.x," 2019.

[75] S. Kullback and R. A. Leibler, "On information and sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[76] C. Angeli *et al.*, "Diagnostic expert systems: From expert's knowledge to real-time systems," *Advanced knowledge based systems: Model, applications & research*, vol. 1, pp. 50–73, 2010.

[77] D. Wang, X. Wang, and Z. Li, "State-based fault diagnosis of discrete-event systems with partially observable outputs," *Information Sciences*, vol. 529, pp. 87 – 100, 2020.

[78] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Springer Science & Business Media, 2009.

[79] C. Seatzu, M. Silva, and J. H. Van Schuppen, *Control of discrete-event systems*, vol. 433. Springer, 2013.

[80] J. Zaytoon and S. Lafortune, "Overview of fault diagnosis methods for discrete event systems," *Annual Reviews in Control*, vol. 37, no. 2, pp. 308 – 320, 2013.

[81] E. Štrumbelj and I. Kononenko, "Explaining prediction models and individual predictions with feature contributions," *Knowledge and Information Systems*, vol. 41, no. 3, pp. 647–665, 2014.

[82] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should I trust you?": Explaining the predictions of any classifier," *CoRR*, vol. abs/1602.04938, 2016.

[83] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and Intelligent Laboratory Systems*, vol. 2, no. 1, pp. 37 – 52, 1987. Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists.

[84] K. Fukuaga, "Introduction to statistical pattern classification," *Patt. Recognit*, vol. 30, no. 7, pp. 1145–1149, 1990.

[85] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[86] T. W. Rauber, F. de Assis Boldt, and F. M. Varejão, "Heterogeneous feature models and feature selection applied to bearing fault diagnosis," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 1, pp. 637–646, 2015.

[87] P. Foroughi, W. Shao, F. Brockners, and J.-L. Rougier, "DESTIN: Detecting state transitions in network elements," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 161–169, 2021.

[88] Y.-L. Lee, P.-K. Tsung, and M. Wu, "Techology trend of edge ai," in *2018 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pp. 1–2, IEEE, 2018.

[89] Z. Duan, Z. Yang, R. Samoilenko, D. S. Oza, A. Jagadeesan, M. Sun, H. Ye, Z. Xiong, G. Zussman, and Z. Kostic, "Smart city traffic intersection: Impact of video quality and scene complexity on precision and inference," in *2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, pp. 1521–1528, IEEE, 2021.

[90] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, 2015.

[91] Y. Gong, L. Liu, M. Yang, and L. D. Bourdev, "Compressing deep convolutional networks using vector quantization," *CoRR*, vol. abs/1412.6115, 2014.

[92] G. Hinton, O. Vinyals, J. Dean, *et al.*, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, vol. 2, no. 7, 2015.

[93] Y. Han, X. Wang, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *arXiv*, vol. 22, no. 2, pp. 869–904, 2019.

[94] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[95] X. Chen, J. Zhang, B. Lin, Z. Chen, K. Wolter, and G. Min, "Energy-efficient offloading for dnn-based smart iot systems in cloud-edge environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 3, pp. 683–697, 2022.

[96] M. Xue, H. Wu, G. Peng, and K. Wolter, "Ddpqn: An efficient dnn offloading strategy in local-edge-cloud collaborative environments," *IEEE Transactions on Services Computing*, vol. 15, no. 2, pp. 640–655, 2022.

[97] T. Mohammed, C. Joe-Wong, R. Babbar, and M. D. Francesco, "Distributed inference acceleration with adaptive dnn partitioning and offloading," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pp. 854–863, 2020.

[98] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive dnn surgery for inference acceleration on the edge," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 1423–1431, IEEE, 2019.

[99] J. Li, W. Liang, Y. Li, Z. Xu, X. Jia, and S. Guo, "Throughput Maximization of Delay-Aware DNN Inference in Edge Computing by Exploring DNN Model Partitioning and Inference Parallelism," *IEEE Transactions on Mobile Computing*, pp. 193–200, 2021.

[100] E. Li, Z. Zhou, and X. Chen, "Edge intelligence: On-demand deep learning model co-inference with device-edge synergy," in *Proceedings of the 2018 Workshop on Mobile Edge Communications*, pp. 31–36, 2018.

[101] H.-J. Jeong, H.-J. Lee, C. H. Shin, and S.-M. Moon, "Ionn: Incremental offloading of neural network computations from mobile devices to edge servers," in *Proceedings of the ACM Symposium on Cloud Computing*, pp. 401–411, 2018.

[102] S. Russell and P. Norvig, *Artificial intelligence: a modern approach, Fourth Edition*. Prentice Hall, 2020.

[103] P. Ongsulee, "Artificial intelligence, machine learning and deep learning," *International Conference on ICT and Knowledge Engineering*, pp. 1–6, 2018.

[104] L. Floridi and M. Chiriatti, "GPT-3: Its nature, scope, limits, and consequences," *Minds and Machines*, vol. 30, no. 4, pp. 681–694, 2020.

[105] A. Munoz, "Machine learning and optimization," 2014.

[106] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[107] I. H. Sarker, "Machine Learning: Algorithms, Real-World Applications and Research Directions," *SN Computer Science*, vol. 2, no. 3, pp. 1–21, 2021.

[108] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[109] A. G. Ivakhnenko, A. G. Ivakhnenko, V. G. Lapa, and V. G. Lapa, *Cybernetics and forecasting techniques*, vol. 8. American Elsevier Publishing Company, 1967.

[110] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[111] I. Aizenberg, N. Aizenberg, and J. Vandewalle, *Multi-Valued and Universal Binary Neurons: Theory, Learning and Applications*. Springer US, 2000.

[112] Z. Li, Y. Zhang, J. Wang, and J. Lai, "A survey of FPGA design for AI era," *Journal of Semiconductors*, vol. 41, p. 021402, feb 2020.

[113] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, pp. 1–12, 2017.

[114] ARM, "Powering the edge: Driving optimal performance with the Ethos-N77 NPU," tech. rep., ARM, 2020.

[115] A. Ignatov, R. Timofte, W. Chou, K. Wang, M. Wu, T. Hartley, and L. V. Gool, "AI benchmark: Running deep neural networks on android smartphones," *CoRR*, vol. abs/1810.01109, 2018.

[116] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, pp. 1–14, 2016.

[117] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge Distillation: A Survey," *International Journal of Computer Vision*, vol. 129, no. 6, pp. 1789–1819, 2021.

[118] C. Yang, L. Xie, C. Su, and A. L. Yuille, "Snapshot distillation: Teacher-student optimization in one generation," *CoRR*, vol. abs/1812.00123, 2018.

[119] S. Yun, J. Park, K. Lee, and J. Shin, "Regularizing class-wise predictions via self-knowledge distillation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 13876–13885, 2020.

[120] L. Zhang, J. Song, A. Gao, J. Chen, C. Bao, and K. Ma, "Be your own teacher: Improve the performance of convolutional neural networks via self distillation," 2019.

[121] Y. Zhang, T. Xiang, T. M. Hospedales, and H. Lu, "Deep mutual learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4320–4328, 2018.

[122] L. L. Zhang, S. Han, J. Wei, N. Zheng, T. Cao, Y. Yang, and Y. Liu, "Nn-Meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices," *MobiSys 2021 - Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 81–93, 2021.

[123] L. Dudziak, T. Chau, M. S. Abdelfattah, R. Lee, H. Kim, and N. D. Lane, "BRP-NAS: Prediction-based NAS using GCNs," *Advances in Neural Information Processing Systems*, vol. 2020-Decem, no. NeurIPS, pp. 1–11, 2020.

[124] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *CoRR*, vol. abs/1612.08242, 2016.

[125] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[126] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-international conference on neural networks*, vol. 4, pp. 1942–1948, IEEE, 1995.

[127] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems," *Econometrica*, vol. 28, no. 3, pp. 497–520, 1960.

[128] A. J. Ferrer, J. M. Marquès, and J. Jorba, "Towards the decentralised cloud: Survey on approaches and challenges for mobile, ad hoc, and edge computing," *ACM Comput. Surv.*, vol. 51, jan 2019.

[129] C. Shi, L. Chen, C. Shen, L. Song, and J. Xu, "Privacy-aware edge computing based on adaptive dnn partitioning," in *2019 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, 2019.

[130] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 595–608, 2021.