TU WIEN Informatics

Aufspaltung neuronaler Netze für energieeffiziente Edge-KI

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Daniel May, BSc
Matrikelnummer 11809922

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof.in Mag.a rer.soc.oec. Dr.in rer.soc.oec. Ivona Brandić
Mitwirkung: Shashikant Shankar Ilager, M.Tech. PhD

Wien, 22. November 2024

Daniel May	Ivona Brandić

TU WIEN Informatics

Neural network splitting for energy-efficient Edge-AI

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Business Informatics

by

Daniel May, BSc
Registration Number 11809922

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof.in Mag.a rer.soc.oec. Dr.in rer.soc.oec. Ivona Brandić
Assistance: Shashikant Shankar Ilager, M.Tech. PhD

Vienna, November 22, 2024

Daniel May | Ivona Brandić

Erklärung zur Verfassung der Arbeit

Daniel May, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 22. November 2024

Daniel May

Danksagung

Ich möchte meiner Betreuerin, Professorin Ivona Brandić, meinen aufrichtigen Dank dafür aussprechen, dass sie mich als ebenbürtiges Mitglied in ihre Forschungsgruppe aufgenommen hat und mir während meiner gesamten Arbeit unschätzbare Hilfe und Unterstützung zukommen ließ. Ich bin meinem Co-Betreuer, Shashikant Ilager, sehr dankbar, der mich von der anfänglichen Themenwahl bis zur Planung der nächsten Schritte und der Konzeption von Experimenten kontinuierlich beraten hat. Seine Verfügbarkeit für Besprechungen und sein Engagement, mir zu helfen, das große Ganze im Blick zu behalten, waren von unschätzbarem Wert. Ich bin auch Alessandro Tundo dankbar, dass er stets für Diskussionen über Detailfragen zur Verfügung stand und immer bereit war zu helfen. Dank der Zusammenarbeit und der Unterstützung dieser drei Personen war ich in der Lage, meine Arbeit bei einer Konferenz einzureichen, wofür ich unendlich dankbar bin.

Mein Dank gilt auch allen anderen Mitgliedern der HPC-Forschungsgruppe, die mir immer wieder ihre Hilfe angeboten haben, sei es durch wertvolles Feedback und Kritiken zu meiner wissenschaftlichen Arbeit oder durch Hilfe im Labor. Ihre Unterstützung in beiden Bereichen habe ich sehr zu schätzen gewusst.

Ich möchte mich bei meiner Familie bedanken, insbesondere bei meiner Mutter, die mich während meines gesamten Studiums unterstützt hat. Ebenso dankbar bin ich meinen Freunden, die mein Leben während meiner Zeit an der Universität bereichert haben. Schließlich möchte ich meiner Partnerin, deren unglaubliches Verständnis und Unterstützung mir in stressigen Zeiten geholfen haben, meine herzliche Anerkennung aussprechen.

Die in dieser Arbeit vorgestellten Experimente wurden mit Hilfe des Grid'5000-Testbeds durchgeführt, das von einer wissenschaftlichen Interessengruppe unterstützt wird, die von Inria organisiert wird und an der das CNRS, RENATER und mehrere Universitäten sowie andere Organisationen beteiligt sind (siehe `https://www.grid5000.fr`).

Diese Arbeit wurde mit einem netidee-Stipendium gefördert.

Diese Forschung wurde teilweise durch die folgenden Projekte finanziert: Transprecise Edge Computing (Triton), Österreichischer Wissenschaftsfonds (FWF), DOI: 10.55776/P36870; Digital Twin for LoRaWAN Agriculture Systems, Steirische Wirtschaftsförderung (SFG), Ideen!Reich XS 1.000.073.260.

Acknowledgements

My deepest appreciation goes to Professor Ivona Brandić, who accepted me into her research team as an equal and offered essential guidance and support throughout my thesis journey. I am tremendously grateful to my co-advisor, Shashikant Ilager, who was consistently present from the moment I chose my topic through to planning future steps and experimental design. His readiness for meetings and commitment to helping me keep the overarching goals in perspective were irreplaceable. I also thank Alessandro Tundo for being persistently available for discussions on detailed issues and his constant willingness to assist. The combined collaboration and support of these three individuals enabled me to submit my work to a conference, for which I am deeply grateful.

I am also grateful to everyone in the HPC Research Group for their unfailing support, be it through offering critical insights and critiques on my academic writing or aiding me in the laboratory. Their help in these aspects has been greatly appreciated.

My deepest gratitude goes to my family, with particular recognition to my mother, for her support during my academic journey. I am also grateful to my friends, whose presence has greatly improved my university experience. Lastly, my sincere thanks go to my partner, whose remarkable patience and encouragement have been invaluable during challenging periods.

Experiments presented in this thesis were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see `https://www.grid5000.fr`).

This work was supported by a netidee scholarship.

This research has been partially funded through the projects: Transprecise Edge Computing (Triton), Austrian Science Fund (FWF), DOI: 10.55776/P36870; Digital Twin for LoRaWAN Agriculture Systems, Steirische Wirtschaftsförderung (SFG), Ideen!Reich XS 1.000.073.260.

Kurzfassung

Der Einsatz von KI-Modellen auf ressourcenbeschränkten Edge-Geräten wird durch begrenzte Rechenkapazitäten und hohen Energiebedarf erschwert. Split Computing bietet eine Lösung, bei der große neuronale Netze aufgeteilt werden und eine partielle Berechnung sowohl auf Edge- als auch auf Cloud-Geräten ermöglicht wird, wobei ein Gleichgewicht zwischen Energieeffizienz und Latenzanforderungen angestrebt wird. Die Bestimmung der optimalen Split-Ebene und Hardwarekonfigurationen ist jedoch nicht trivial. Diese Komplexität ergibt sich aus dem großen Konfigurationsraum, nichtlinearen Abhängigkeiten zwischen Software- und Hardwareparametern, heterogenen Hardwareeigenschaften und dynamischen Lastbedingungen.

Um diese Herausforderungen zu bewältigen, schlagen wir `DynaSplit` vor, ein umfassendes zweistufiges Hardware-Software-Optimierungsmodell. `DynaSplit` konfiguriert dynamisch sowohl Software-Parameter (d.h. die Split-Ebene) als auch Hardware-Einstellungen (z.B. Beschleunigernutzung, CPU-Frequenz) zur Leistungsoptimierung. In der *Offline Phase* gehen wir das Problem mit einem multikriteriellen Optimierungsansatz an, indem wir einen meta-heuristischen Algorithmus nutzen, um Pareto-optimale Konfigurationen zu finden. In der *Online-Phase* identifiziert ein Scheduling-Algorithmus die am besten geeigneten Einstellungen für jede eingehende Inferenzanfrage, um einen minimalen Energieverbrauch zu gewährleisten und gleichzeitig die durch die Quality of Service (QoS)-Vorgaben der Anwendung definierten Latenzanforderungen zu erfüllen.

Unsere Implementierung von `DynaSplit`, die an einem realen Prototyp unter Verwendung gängiger vortrainierter KI-Modelle getestet wurde, zeigt erhebliche Energieeinsparungen und Leistungsverbesserungen. Experimentelle Ergebnisse zeigen, dass `DynaSplit` den Energieverbrauch im Vergleich zu reinen Cloud-Berechnungen um bis zu 72% reduzieren kann und ca. 90% der benutzerspezifischen Latenzanforderungen erfüllt, was eine deutliche Überlegenheit gegenüber herkömmlichen Baselines darstellt.

Abstract

The use of artificial intelligence (AI) models on edge devices with limited resources faces obstacles due to insufficient computational capacity and excessive energy consumption. Split computing addresses this issue by dividing neural networks (NNs) so that some computations occur on edge devices and some in the cloud. This approach manages energy efficiency and latency demands. However, finding the best layer to split and correct hardware setups is complex. This difficulty is due to a vast array of configurations, non-linear interactions between software and hardware, diverse hardware features, and fluctuating workload scenarios.

In response to these hurdles, we introduce `DynaSplit`, an extensive framework to optimize hardware and software in two distinct phases. `DynaSplit` dynamically adjusts software components, such as the split layer, along with hardware configurations such as accelerator usage and CPU frequency, to enhance performance. During the *Offline Phase*, we tackle the optimization issue employing a multi-objective approach with a meta-heuristic algorithm to find Pareto-optimal setups. Meanwhile, the *Online Phase* employs a scheduling algorithm to select the optimal settings for every incoming inference task, thereby minimizing energy usage while adhering to the latency thresholds imposed by the application's quality of service (QoS) constraints.

By deploying `DynaSplit` on a real-world prototype with widely-used pre-trained AI models, we achieved notable energy efficiency while meeting application requirements. Our experimental data indicate that `DynaSplit` can reduce energy usage by as much as 72% in contrast to cloud-only solutions and can meet around 90% of user-defined latency targets, thus greatly exceeding baselines.

Contents

CHAPTER 1

Introduction

This thesis addresses the challenges of deploying resource-intensive deep neural networks (DNNs) in constrained environments, focusing on energy-efficient, latency-aware solutions through split computing and hardware-software co-design.

1.1 Context & Motivation

Deep neural networks (DNNs) have revolutionized the field of artificial intelligence (AI), driving advancements in areas such as image classification, natural language processing, and autonomous systems [ZMB+21, RN20]. These models are inspired by the human brain and consist of interconnected layers of artificial neurons designed to identify patterns and make predictions from data [GBC16]. During the past decade, improvements in computational power, data availability, and algorithm design have enabled DNNs to achieve state-of-the-art performance on numerous tasks [YHPC18, AGH20]. Their ability to process complex data and generate accurate insights has made them indispensable in modern AI applications.

Despite their remarkable capabilities, DNNs are computationally intensive and require substantial hardware resources for both training and inference [HTR22, DBMC22]. Although the inference phase, where a trained model is executed to make predictions, is less demanding than training, it remains resource intensive due to the size and complexity of modern models [BCCN18]. This is particularly critical because models are typically trained once but widely deployed, with inference operations occurring repeatedly and at scale, especially in publicly available AI services. The frequent and widespread use of inference amplifies its resource demands, leading to significant energy consumption and contributing to growing concerns about the environmental impact of AI technologies. Studies show that the carbon footprint of deploying DNNs, particularly in large-scale cloud infrastructures, can be substantial [SDSE20, SGM19]. These challenges underscore the need for energy-efficient deployment strategies.

Edge applications have become increasingly intelligent, using advanced artificial intelligence algorithms to enable real-time decision making and processing [WHL+20, LMP+21, LLW19, SSWP21]. This trend, referred to as *Edge AI*, allows intelligent applications at the edge. It involves deploying DNNs on edge devices such as smartphones, IoT sensors, and embedded systems. However, deploying these models on edge devices is challenging due to their resource-constrained nature, with limited computational power, memory, and energy budgets [WHL+20, CJLF16, CLM+21]. Unlike centralized cloud data centers, which offer scalable resources [AFG+10], edge devices must handle inference tasks in real time while minimizing latency and energy usage. This makes direct deployment of resource-intensive DNNs on edge devices infeasible for many applications. For example, autonomous vehicles and real-time healthcare monitoring require ultra-low latency to ensure timely responses, yet these applications often involve complex models that exceed the capabilities of edge hardware [PCT+20].

To address the limitation of deploying DNNs on constrained hardware, *split computing* has emerged as a promising approach. Split computing involves partitioning a DNN into segments that are executed across different devices, such as edge and cloud resources [MLR23, KHG+17]. For example, the initial layers of a network might be processed on the edge device, capturing and compressing raw data into intermediate features, while the remaining layers run on the cloud, leveraging its computational power. By distributing the computation, split computing can balance the trade-offs between latency, energy consumption, and inference accuracy [LFEF24, ZZL+24, MTIB24].

However, achieving optimal performance in split computing is not straightforward. The choice of split point, determining which layers run on the edge versus the cloud, has a significant impact on latency and energy consumption [KHG+17]. This relationship is highly non-linear and is influenced by factors such as the size of intermediate outputs, network bandwidth, and the workload characteristics of edge devices and cloud devices [EAP21]. Moreover, existing solutions in split computing often fail to incorporate hardware parameter tuning into their optimization processes. Parameters such as accelerator usage (TPU, GPU) and dynamic voltage and frequency scaling (DVFS) can drastically affect energy efficiency and latency [TWWC19, HMS21], but are typically overlooked in split computing frameworks.

Although some approaches dynamically adapt to runtime variations and QoS requirements, they often optimize only the software-level configurations, neglecting the potential benefits of a combined hardware-software co-design. Addressing this gap, this thesis proposes a comprehensive framework that integrates hardware parameter tuning into split computing optimization. By jointly considering hardware and software interactions, the framework aims to provide energy-efficient DNN inference while meeting strict latency and accuracy requirements.

1.2 Research Problem & Objectives

The deployment of DNNs on edge devices is challenging due to their high computational and energy demands. Edge devices often lack the resources required for efficient inference, making it difficult to balance performance, energy efficiency, and accuracy.

Split computing offers a potential solution by dividing the computational workload between edge devices and the cloud. This approach reduces the resource burden on the edge device while leveraging the computational power of the cloud. However, selecting the optimal split point is challenging because DNN computations do not scale linearly across layers, and their performance depends both on the software parameters, such as the NN architecture, and hardware capabilities, such as processing power and memory.

In addition to split selection, hardware configurations such as CPU frequency, GPU utilization, and network conditions play a crucial role in determining the energy consumption and latency of inference. The dynamic nature of runtime conditions, combined with strict QoS requirements like latency deadlines, further complicates the optimization process.

This work aims to address these challenges by developing an energy-aware framework for split computing. The objectives are as follows.

- Identify critical parameters that influence the energy consumption and latency of DNN inference.
- Determine optimal split points to divide computation between edge devices and the cloud.
- Dynamically configure hardware parameters to meet QoS requirements while ensuring energy efficiency.

The research questions addressed in this thesis are as follows:

RQ_1: What are the appropriate strategies for splitting DNNs used in image classification?

RQ_2: How do hardware parameters influence the trade-offs between energy consumption, accuracy, and latency in DNN inference?

RQ_3: How can we jointly optimize model splitting and hardware configurations to meet energy and latency requirements for DNN inference?

A preliminary version of this work has been published as a preprint [MTIB24], which forms the basis for this thesis. Building on this previous work, the thesis provides an expanded background, more detailed analyses, and refined methodologies to address the stated research questions.

1.3 Methodological Approach

This thesis uses a structured methodology to address the research questions presented, integrating a systematic review of the literature, a motivational study, the development of a software artifact and comprehensive evaluations. The approach ensures a consistent narrative, from understanding the problem space to delivering and validating a solution.

1.3.1 Literature Review: Establishing a Foundation

The first step involves conducting a systematic literature review inspired by the methodology of Kitchenham and Charters [KC07]. This review focuses on strategies for splitting deep neural networks (DNNs) in image classification tasks and their impact on energy efficiency and latency. Using well-defined search queries across major digital repositories, such as the ACM Digital Library, IEEE Xplore, and Google Scholar, this review seeks to identify existing approaches, challenges, and knowledge gaps. Special attention is paid to techniques that balance trade-offs between energy consumption, latency, and accuracy, as these are crucial to optimizing DNN inference in resource-constrained edge environments. The insights gained from this review directly address RQ_1 by identifying appropriate DNN splitting strategies.

1.3.2 Motivational Study: Empirical Insights on Hardware Parameters

Based on the literature review, a motivational study will be conducted to provide empirical evidence on how various hardware configurations influence energy consumption, latency, and accuracy during DNN inference. This study bridges the gap between theory and practical deployment, focusing on:

- *Accelerators:* Analyzing the performance of GPUs and TPUs under varying workloads.
- *DVFS:* Investigating the effects of adjusting CPU and TPU frequencies on inference metrics.
- *Split Points:* Exploring how different neural network splitting configurations affect energy, latency, and accuracy.

Through controlled experiments [GL87] in the RUCON[1] lab at TU Wien, which offers edge devices and energy measurement tools, and the Grid5000[2] research cloud testbed, the study will simulate realistic conditions to identify key parameters that can be optimized in the DNN inference. These findings address RQ_2, laying the groundwork for designing an optimization strategy by identifying tunable parameters and their trade-offs.

[1] http://rucon.ec.tuwien.ac.at
[2] https://www.grid5000.fr/w/Grid5000:Home

1.3.3 Optimization and Artifact Development

The insights from the motivational study will inform the design and implementation of a software artifact, aligned with the design science research framework of Hevner et al. [HMPR04]. This artifact will serve as a prototype system to jointly optimize DNN splitting strategies and hardware configurations. By dynamically selecting the split point and tuning the hardware parameters, the system aims to meet specific energy and latency requirements, addressing RQ_3.

The artifact leverages multi-objective optimization techniques to balance energy efficiency with latency constraints, ensuring robust performance across diverse workloads. The implementation will consider both edge and cloud environments, integrating mechanisms for adaptability and scalability in real-world scenarios.

1.3.4 Evaluation: Comparison against baselines

To validate the effectiveness of the artifact developed, a detailed evaluation will be carried out. This includes:

1. *Quantitative Analysis:* Measuring energy, latency and accuracy as the artifact dynamically selects split points and hardware settings to meet QoS requirements for user requests.

2. *Baseline Comparisons:* Comparing the performance of the artifact against traditional approaches, such as computation based solely on the cloud or edge, under identical conditions.

3. *Scalability Testing:* Simulating high workloads to assess the system's ability to maintain performance under increasing demands.

The RUCON lab and the Grid5000 research cloud testbed provide an ideal testing environment, ensuring reliable measurements and repeatable experiments. These evaluations will highlight the advantages of the proposed joint optimization framework over existing solutions.

This methodical approach forms a coherent workflow: understanding the state-of-the-art, producing empirical evidence, crafting and applying a customized solution, and rigorously testing its effectiveness. This process systematically covers research questions, aiming to improve DNN inference in resource-constrained settings.

1.4 Outline

This thesis is structured to build a clear narrative from the basic concepts to the evaluation and implications of the proposed framework. It begins by laying the foundation and systematically progresses toward addressing the research questions.

The discussion begins in Chapter 2, where the essential foundations for this work are established. The topics include the principles of edge and cloud computing, the basics of AI and ML, and the challenges faced in deploying models on edge devices. This chapter also explores existing techniques for overcoming these challenges, concluding with a detailed introduction to split computing, which serves as the basis for the subsequent chapters.

Building on this groundwork, Chapter 3 places the thesis within a broader research context. Reviews prior work on energy-aware split computing, critically evaluating existing approaches, and identifying their limitations. This analysis highlights the gaps that this thesis aims to address, setting the stage for our contributions.

With the context established, Chapter 4 formalizes the problem addressed in this thesis. Key terms are defined, and the precise formulation of the research problem is detailed, ensuring clarity and a focused direction for the solution.

The core of this thesis is presented in Chapter 5, which introduces the `DynaSplit` framework. This chapter outlines the design of the proposed solution, detailing the optimization strategy employed during the *Offline Phase* and the dynamic configuration methods applied in the *Online Phase* to adapt to user requirements.

To translate the conceptual framework into a practical system, Chapter 6 discusses the implementation of `DynaSplit`. It describes the software setup, addressing the development challenges encountered and the strategies used to overcome them.

The effectiveness of the proposed solution is demonstrated in Chapter 7, which presents a detailed evaluation of the framework. This chapter includes the experimental setup, testbed design, and simulation studies. Key performance metrics, such as energy efficiency and latency, are analyzed and the limitations of the approach are critically assessed to provide a comprehensive understanding of its strengths and areas of improvement.

Finally, Chapter 8 concludes the thesis by summarizing the main findings and describing potential future research directions. The chapter emphasizes the broader impact of this work and its contribution to advancing the field of energy-aware split computing.

This structure ensures a logical progression, connecting foundational insights, theoretical contributions, and practical implementations to provide a holistic overview that addresses research questions and highlights the significance of the `DynaSplit` framework.

CHAPTER 2

Background and Preliminaries

This chapter presents a comprehensive summary of essential principles associated with the `DynaSplit` framework. It explores Cloud and Edge Computing, presents an introduction to artificial intelligence (AI) and its use cases, examines the challenges and solutions of Edge AI, and wraps up by discussing Split Computing as a practical approach for addressing these challenges.

2.1 Cloud and Edge Computing

Cloud computing refers to the provision of computing resources and applications over the Internet [AFG+10]. These resources are categorized into three primary service models:

- *Infrastructure as a Service (IaaS):* Provides computing resources such as servers, storage, and networks, enabling users to deploy and manage their software, operating systems, and applications.
- *Platform as a Service (PaaS):* Offers a platform for developers to build, test, and deploy applications without managing the underlying hardware and software infrastructure.
- *Software as a Service (SaaS):* Delivers software applications over the Internet, allowing users to access them on-demand without the need for installation or maintenance.

These service models are often integrated within cloud computing environments, making their boundaries sometimes indistinct. For simplicity, they are jointly referred to under the umbrella of "cloud computing."

Cloud services are typically centralized and offered through remote data centers, which host the hardware and software infrastructure. The development of large-scale, low-cost, centralized data centers has been a key driver in the rise of cloud computing, enabling significantly reduced costs and higher efficiency compared to traditional decentralized IT systems [AFG+10]. Public cloud solutions, accessible to the general public, leverage economies of scale to further enhance cost-efficiency.

Cloud computing offers several advantages due to its centralized architecture [AFG+10, BVS13]:

- *Scalability*: The cloud allows for smooth resource scaling throughout the computing stack, adapting to workload variations without needing physical infrastructure investments.
- *Cost Efficiency*: Cloud computing shifts the IT infrastructure to a pay-as-you-go model, lowering upfront investments and maintenance costs by letting the cloud service providers handle the infrastructure.
- *On-Demand Access and Flexibility*: Users can access cloud resources on-demand, enabling rapid scaling to meet demand without prior commitments.
- *Global Accessibility*: Cloud services can be accessed from anywhere and any device with an internet connection, enhancing collaboration and productivity.

The cloud market is dominated by some vendors such as Amazon Web Services, Microsoft Azure and Google Cloud, which hold the highest market share in the cloud market, which globally reported revenue in Q3 2024 of more than $84 billion [SRG24]. This infrastructure has enabled transformative applications in numerous domains, including real-time health monitoring systems that support remote diagnostics, large-scale satellite image processing for environmental and disaster analysis, and powerful data analytics tools that accelerate scientific research [BVS13]. These advancements highlight the role of the cloud in driving innovation by making high-performance computing accessible and adaptable to a wide range of fields.

As cloud computing drives innovation with centralized, high-performance resources, the demand for local processing has led to the development of edge computing. Edge computing is a paradigm aimed at performing data processing close to its origin [SCZ+16]. Unlike traditional cloud computing, which depends on centralized data centers, edge computing allows devices or nodes near the data source (such as sensors, cloudlets, or gateways) to process data directly. This approach effectively minimizes latency, reduces bandwidth usage, and improves data privacy, particularly in Internet of Things (IoT) and mobile computing situations [Sat17].

Edge computing offers several advantages over centralized cloud computing, as described in [SCZ+16, Sat17]:

- *Reduced Latency*: Processing data closer to its source minimizes network delays, which is essential for time-sensitive applications such as augmented reality (AR) and autonomous driving.

- *Optimized Bandwidth Use*: By decreasing the need to send extensive data streams to the cloud, edge computing conserves bandwidth, especially useful in high-data scenarios such as video analytics and IoT sensor networks.

- *Enhanced Privacy and Security*: Local processing allows sensitive data to be filtered or anonymized before it reaches the cloud, offering better privacy control.

- *Reliability and Resilience*: With edge devices capable of functioning independently from the cloud, systems become more robust against network disruptions.

With the advantage of reduced latency, edge computing enables innovative applications in fields such as autonomous vehicles and real-time video analytics. Localized processing can also improve energy efficiency, especially for battery-operated IoT devices, by minimizing data transfer demands [SCZ+16]. Furthermore, collaborative edge networks allow multiple edge nodes to share resources and insights, enhancing both robustness and scalability in systems such as smart cities [SCZ+16]. However, challenges remain, including ensuring data privacy, achieving interoperability between various devices, and managing the limited resources of distributed edge nodes [Sat17].

After examining cloud and edge computing, it becomes crucial to acknowledge the concept of the edge-cloud continuum. This idea introduces a range of computational resources that spans from end-user devices to central cloud data centers. Instead of simply classifying resources as cloud or edge, this continuum offers various tiers, starting from the device edge where user devices handle computations, advancing through the gateway edge and regional data centers, to remote cloud data centers [AZTS18, YLH+18]. Distinctive features, user proximity, and inherent resource constraints characterize each level within this spectrum, thereby facilitating more refined and adaptable computing strategies. In Figure 2.1 we can see an example visualization of this continuum, showcasing the hierarchical organization. It shows the inner-edge, middle-edge, and outer-edge layers, highlighting their roles and the typical devices associated with each layer.

The edge-cloud continuum aims to enable more precise trade-offs among performance objectives such as latency, energy efficiency, and computing power. To illustrate, while the device edge minimizes latency, its processing capabilities are often constrained. Conversely, cloud data centers deliver substantial processing strength but experience increased latency due to their remote location from the user. By utilizing intermediate tiers, like the gateway edge and regional data centers, tasks can be allocated dynamically in real time, optimizing for low latency, energy conservation, or heightened processing capacity as required [YLH+18]. This stratified strategy allows for a wide array of trade-offs that can be customized to meet the specific needs of each application.

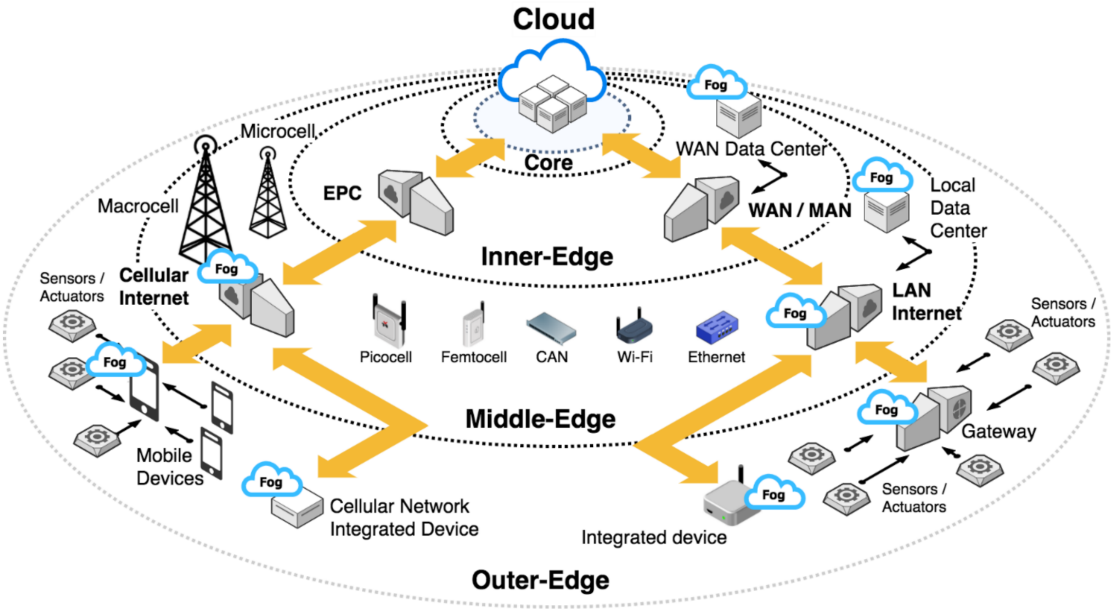


Figure 2.1: An example of the layered architecture of the edge-cloud continuum as shown by [CSB19].

This continuum-based approach proves to be particularly advantageous for applications in real-time Internet of Things (IoT), augmented reality (AR), and connected vehicles. These scenarios experience varying demands for low latency, energy efficiency, and processing power. For instance, in AR, data processing that occurs instantly at the Gateway Edge is crucial for ensuring quick interactions. Meanwhile, analytics that are not immediately pressing can be performed further along the continuum. This method facilitates the effective allocation of tasks and enhances the overall user experience [AZTS18, YLH+18].

In summary, cloud and edge computing offer complementary benefits: cloud computing provides scalable, cost-effective resources but can suffer from high latency, while edge computing reduces latency by processing data locally, although with limited resources [AFG+10, SCZ+16]. The edge-cloud continuum integrates both approaches, spanning layers from the device edge to cloud data centers, allowing dynamic trade-offs between latency, energy, and computational power [AZTS18, YLH+18]. This layered model supports various applications such as IoT and AR, where efficient and adaptable computing is essential.

2.2 Artificial Intelligence

The domain of artificial intelligence (AI) aims at the creation of systems capable of replicating human intelligence in order to undertake tasks such as perception, decision-making, and problem-solving [RN20]. AI integrates a variety of methodologies, ranging from traditional rule-based systems to modern innovations in machine learning (ML)

and deep learning (DL). This enables computers to process intricate and frequently unstructured data in a manner that seems intelligent to humans. Additionally, AI encompasses knowledge representation and automated reasoning, which empowers systems to deduce new insights and make informed decisions utilizing available data [RN20].

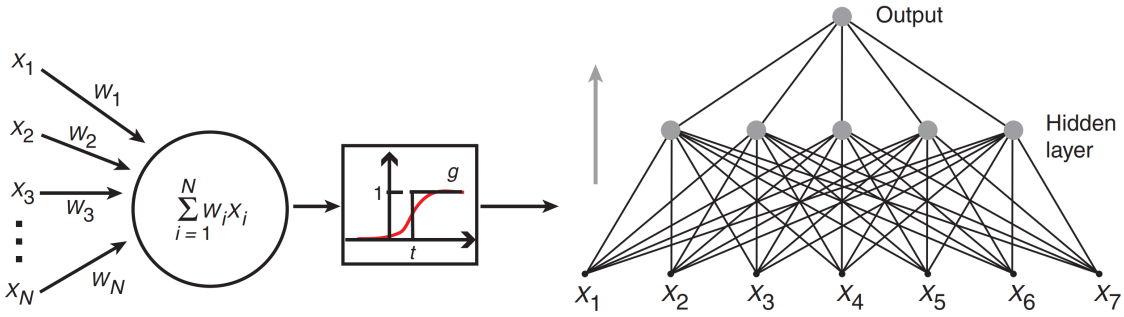
The field of machine learning (ML) represents a specialized area within AI, dedicated to the development of algorithms that can learn from data and make predictions. Unlike traditional approaches that rely on predefined rules, ML systems adjust and evolve by identifying patterns within the data, improving their performance as they encounter new datasets [RN20]. The function of ML is to enable systems to acquire data representations, which are crucial for activities such as classification and regression [GBC16].

Within ML, deep learning (DL) represents a specialized area that takes advantage of neural networks with multiple layers, commonly known as deep neural networks, to analyze and comprehend data. DL models have revolutionized areas such as image recognition, natural language processing, and speech analysis by efficiently managing large datasets with minimal need for manual feature engineering [RN20, GBC16]. DL is characterized as the approach through which a machine learns to conceptualize the world in hierarchical terms, beginning from unprocessed data and gradually revealing increasingly abstract features as data flow through various layers of the network [GBC16].

Although the general objective of AI is to realize intelligent behavior, it is ML that acts as the means to learn from data, while DL employs a neural network strategy with multiple layers, essential for handling high-dimensional data. This differentiation naturally leads us to the importance of neural networks, which serve as the basis for DL models.

Neural networks serve as computational models that draw inspiration from the complex interconnected structure of human brain neurons. A fundamental model in this field was brought forward by McCulloch and Pitts in 1943, where they conceptualized the neuron as a “threshold unit”. This neuron, depicted in Figure 2.2a, is designed to receive input from various sources, with each input x_i having a corresponding weight w_i associated with it [MP43]. The neuron computes a weighted sum, denoted as $\sum_{i=1}^{N} w_i x_i$. If this resulting sum exceeds a certain pre-established threshold t, the neuron generates an output of 1, if not, the output remains 0 [Sch15]. Subsequently, this threshold model was enhanced by integrating smoother activation functions, such as the sigmoid function. This integration enables the production of continuous outputs and enables training using gradient-based methods [Sch15].

Modern neural networks are constructed around this principle by structuring neurons across several distinct layers, namely: the input layer, hidden layers, and the output layer. Take, for example, a basic feed-forward network depicted in Figure 2.2b. It possesses an input layer receiving data, one or more hidden layers that process the data through connections that are weighted accordingly, and an output layer that yields the final result. Within each hidden layer, an activation function, such as Rectified Linear Unit (ReLu), is employed. This function integrates non-linearity into the network, thus allowing it to recognize and comprehend complex patterns present within the data [RN20].



(a) McCulloch-Pitts neuron model (b) Feed-forward network

Figure 2.2: Basic neural network concepts [Kro08].

Networks distinguished by their numerous hidden layers are called deep neural networks (DNNs). They enable intricate data representations and serve as the basis for complex AI applications. Different variants, including convolutional neural networks (CNNs), are particularly effective in handling image processing tasks. Recurrent neural networks (RNNs) are tailored for analyzing sequential data, while transformer architectures have brought significant advances to natural language processing [Sch15, VSP+17]. These deep network structures serve as the basis for a range of AI systems, highlighting their important role in various technological domains.

AI systems have become essential tools in a wide range of applications. In the realm of classification, DNNs are utilized to categorize various types of data, such as organizing emails or diagnosing medical images. Object detection, which integrates classification with the ability to locate objects, is widely used in both autonomous vehicles and surveillance systems, allowing the real-time identification and tracking of objects [GDDM14]. In the field of natural language processing, AI models play a crucial role in understanding and generating human language, thus empowering applications such as machine translation and sentiment analysis [VSP+17]. In addition, speech recognition systems capitalize on these networks to convert spoken language into text with high accuracy. Meanwhile, recommendation systems customize content for users based on their activity, increasing user engagement on digital platforms [YHC+18]. Together, these diverse applications highlight the broad applicability and significant influence of AI-driven technologies in our daily lives.

The extensive use of AI models, particularly DNNs with their complex architectures, results in substantial energy consumption during both the training and inference stages. In particular, the training phase is notably energy-intensive as it typically involves processing large datasets and managing millions of parameters. As an example, recent research indicates that training a single deep learning model can emit as much CO_2 as five cars would over their entire lifetime, highlighting the environmental impact of modern AI research [SGM19]. Furthermore, Schwartz et al. propose the idea of "Green AI", stressing the need to focus on creating energy-efficient AI models and strategies to reduce environmental issues arising from large-scale training [SDSE20]. The energy

demands are even more pronounced in large and cutting-edge models, which requires vast computational resources and infrastructure [SGM19].

The substantial energy requirements related to the training and inference processes in AI highlight the importance of focusing on efficiency during model design and deployment. Taking actions to mitigate this energy footprint is vital not only from an environmental responsibility point of view, but also as AI systems become increasingly widespread. In summary, tackling these energy issues will become even more critical as AI integrates into environments such as edge computing, where resource constraints highlight the need for sustainable AI deployment strategies.

2.3 Edge AI: Opportunities and Deployment Challenges

This section explores the opportunities, challenges, and hardware factors influencing Edge AI deployment.

2.3.1 Opportunities of Edge AI

Edge AI refers to the integration of AI with edge computing, involving the use of AI methodologies on edge devices to provide localized intelligence, improve response time, and maintain data privacy [DPM+22]. This approach facilitates a novel category of intelligent applications capable of functioning autonomously by performing data analysis proximate to the data source. Thus, an increasing number of edge applications are integrating artificial intelligence (AI) features directly into edge devices, improving the data analysis capabilities at the edge of the network [LMP+21, LLW19, SSWP21]. Edge AI applications occur in a diverse range of fields, such as autonomous driving, healthcare, smart manufacturing, and environmental monitoring, where the capacity for real-time data analysis and prediction is crucial to system functionality and responsiveness [WHL+20, DZF+20]. In these scenarios, incorporating machine learning (ML) models enables information to be processed locally on devices, thus minimizing latency and improving privacy by limiting the need to transfer data to centralized cloud servers.

Implementing AI functionalities at the edge allows faster actions and insights in scenarios where real-time decisions are paramount [WHL+20]. Moreover, edge AI can decrease network load, thus reducing the need for bandwidth and cutting down operational expenses by shifting computational tasks from centralized cloud infrastructures to nearby edge nodes [DZF+20].

2.3.2 Challenges in Edge AI Deployment

Deploying AI models on the edge presents several difficulties due to typical constraints of the edge environment. In particular, edge nodes are generally equipped with fewer computational resources, such as less processing power, insufficient memory, and storage, compared to cloud servers [WHL+20, CJLF16, CLM+21]. This poses a significant challenge for AI models, which are increasingly complex and demanding in terms of

resources, especially deep neural networks (DNNs) that comprise millions of parameters. As a result, edge AI needs to manage the computational requirements for model inference while dealing with the resource limitations present at the edge.

Power limitations and potential unreliability, particularly in remote or mobile settings, add to the resource constraints at the edge [WHL+20]. To maintain reliable operation, energy-efficient inference methods designed specifically for edge devices must be developed [CLM+21]. Overcoming these constraints requires optimizing hardware and software to improve the performance and efficiency of edge AI applications. This optimization is crucial to the successful deployment of AI at the edge, addressing the inherent resource limitations of the edge nodes for effective and scalable AI use in real settings [DZF+20].

Furthermore, the workloads on edge devices are dynamic, with users often requesting varying QoS in terms of latency and quality of inference [ZWZ+23]. This variability requires satisfying diverse service-level agreement (SLA) requirements, further complicating the optimization process for effective and scalable AI deployment in real-world scenarios.

2.3.3 System-Level Factors Influencing Edge AI Performance

The deployment of AI models on edge devices is shaped by hardware choices and configurations that affect energy efficiency, latency, and inference accuracy. This section explores the role of edge AI accelerators, trends in their performance, and hardware configuration knobs used to optimize resource use.

Edge AI accelerators, such as Google's Coral Edge TPU, NVIDIA's Jetson Nano, and Apple's Neural Engine, are designed to perform DNN inference efficiently under strict power budgets. For example, the Jetson Nano achieves up to 472 GOPS at just 5 to 10 watts, making it suitable for low-power applications [LL20, HMS21]. These accelerators enable real-time tasks like object detection and classification, while reducing reliance on cloud resources. However, as models grow in size, the minimal improvement in accuracy comes at a disproportionately higher energy cost [HMS21].

Recent evaluations of DNN inference on edge accelerators reveal that lightweight models, such as MobileNet and EfficientNet, provide better accuracy per joule than larger architectures such as ResNet50 and VGG [HMS21]. For example, while ResNet50 achieves higher accuracy, its energy consumption makes it less suitable for energy-constrained environments. These findings highlight the need for careful model selection based on application requirements and hardware capabilities.

Beyond model selection, hardware configuration plays a critical role in optimizing edge AI performance. Dynamic voltage and frequency scaling (DVFS), a widely used technique to balance performance and energy efficiency, dynamically adjusts CPU or GPU frequencies dynamically. Studies show that optimizing GPU frequencies can reduce energy consumption by up to 26.4% during inference [TWWC19]. Higher GPU frequencies generally improve latency, but diminishing returns at extreme settings and memory bottlenecks require the finding of an optimal balance [TWWC19, DRL+23].

On devices like NVIDIA's Jetson Nano, tuning the CPU and GPU frequencies together reveals additional energy savings opportunities [DRL+23]. However, these optimizations must account for workload characteristics, as the most energy-efficient configuration often depends on the specific DNN and its computational requirements.

These insights into edge accelerators and hardware configurations underscore their importance in overcoming resource constraints and achieving efficient AI inference. In the next section, we discuss complementary techniques, such as model compression and workload distribution, which further address the challenges of deploying AI at the edge.

2.4 Techniques for Addressing Edge AI Deployment Limitations

To mitigate the challenges of deploying AI models on resource-constrained edge devices, two primary strategies have been developed: improving on-device computation and distributing workloads across multiple devices [CR19]. These approaches aim to optimize the use of available resources while maintaining acceptable performance levels in terms of accuracy, latency, and energy efficiency.

2.4.1 On-Device Computation

When talking about on-device computation, we focus on enabling edge devices to independently perform inference by optimizing AI models to meet the constraints of their hardware and power environments. This includes the development of more efficient model designs and the application of model compression techniques [CR19, MBCM22].

Efficient Model Design

The first approach, efficient model design, prioritizes reducing the computational and memory requirements of AI models without significantly sacrificing their accuracy. A prime example is the MobileNets family of architectures [HZC+17, SHZ+18, HPA+19, QLD+24], which use depthwise separable convolutions and inverted residual blocks to dramatically decrease the number of parameters and floating-point operations (FLOPs). MobileNetV3 further improves efficiency through neural architecture search and lightweight attention modules [HPA+19]. EfficientNet [TL19, TL21], another widely adopted architecture, scales models systematically in terms of depth, width, and resolution to achieve an optimal trade-off between accuracy and efficiency. It also seeks to use hardware-friendly operation blocks to increase performance.

Other architectures, such as YOLO [RDGF16, RF17, RF18, BWL20] and SqueezeNet [IMA+16], are tailored for real-time applications. YOLO started to achieve high-speed object detection by using a single neural network to predict bounding boxes and class probabilities simultaneously. SqueezeNet, on the other hand, uses a "fire module" structure to minimize parameters while maintaining similar accuracy to larger models like AlexNet, making

it ideal for edge inference. These efficient architectures enable edge devices to perform complex tasks like image recognition and object detection in real-time with limited resources.

Model Compression

Model compression techniques improve the feasibility of deploying existing DNN models on edge devices by reducing their size and computational requirements [CR19, MBCM22]. The three most common techniques are:

- *Quantization:* Quantization reduces the precision of model parameters and operations from floating-point (e.g., 32-bit) to lower-precision formats such as 8-bit integers. This not only reduces memory usage, but also enables faster inference by leveraging hardware accelerators optimized for integer arithmetic [JKC+18, HMD16]. Quantization-aware training ensures that the model remains robust despite reduced precision.

- *Pruning:* Pruning removes unnecessary weights or connections in a neural network, effectively reducing its size and computational complexity [HMD16]. Structured pruning, which removes entire neurons or channels, is particularly effective for edge devices, as it aligns well with the requirements of hardware implementations. Pruning can be applied iteratively during training to maintain model accuracy.

- *Knowledge Distillation:* This technique involves training a smaller "student" model to mimic the output of a larger, more complex "teacher" model [HVD15]. The student model learns to replicate the teacher's behavior through a softened probability distribution, capturing the generalization abilities of the larger model. Knowledge distillation is widely used to create compact models that are well-suited for edge deployments without compromising significantly on accuracy.

These model compression techniques can be applied individually or in combination to produce highly efficient models that meet the computational and memory constraints of edge devices. The resulting models are capable of performing inference with minimal energy consumption, enabling sustainable and effective deployment in real-world applications [HMD16, JKC+18, HVD15].

2.4.2 Workload Distribution

In addition to optimizing on-device computation, workload distribution leverages collaborative capabilities of multiple devices to handle computational demands more effectively [CR19]. This strategy includes federated learning [MBCM22] and split learning [CLY+24] during the training phase and split computing [CR19] for inference.

Federated Learning

Federated learning enables edge devices to collaboratively train a global model without sharing raw data [MMR+17]. Each edge device trains a local model on its private dataset and sends only the model updates to a central server, where they are aggregated to update the global model. This decentralized training approach reduces communication overhead and ensures data privacy, making it particularly suitable for sensitive applications like healthcare and finance. By keeping data localized, federated learning addresses privacy concerns while taking advantage of distributed computational resources.

Split Learning

Split learning is a distributed training technique that divides the model into a "head" executed on the client and a "tail" on the server. During training, the client processes data through the head, sends intermediate representations to the server for further computation and backpropagation, and receives gradients to update the local model [CLY+24, SMB22]. This approach reduces communication overhead compared to sending raw data, making it suitable for privacy-sensitive and resource-constrained environments [LWWW24, CLC22, MVO+24].

Unlike Federated Learning, where clients train entire models locally and only share updates with a central server, Split Learning divides the model into two parts. Clients handle the initial layers and send intermediate activations to a server, which completes the remaining computation [SMB22, CLC22]. This approach reduces the computational burden on resource-constrained clients, making it suitable for devices such as IoT systems. However, unlike parallel training in Federated Learning, Split Learning relies on sequential communication between clients and the server, which can limit scalability in environments with many clients.

Split Computing

Split computing, used during inference, allocates AI model execution tasks between edge devices and also external resources such as cloud servers [MLR23, KHG+17]. There are various approaches to divide neural networks and place it on different devices. This is especially useful for large models, optimizing resource use, latency, and energy consumption, and allowing advanced AI models on edge devices beyond their capacity limits. The next section explores split computing further, focusing on giving an overview.

2.5 Split Computing

Split computing partitions the execution of DNN models between different devices. By processing initial layers locally and offloading subsequent layers, split computing reduces latency, optimizes resource utilization, and enables resource-constrained devices to run advanced AI models [MLR23]. This approach has been applied to tasks such as image classification [KHG+17, LOD18, JJLM18], object detection [CB18, CCB20],

and sentiment analysis [PCMP21], using models such as VGG16 [LLW+18, INY21], ResNet50 [EEP19, CCB20], and GoogLeNet [JJLM18, LOD18].

Split computing approaches can be categorized into two types:

- *Without Architectural Modifications:* These methods exploit natural bottlenecks within model architectures, which are intermediate layers with an output smaller than the input, to identify split points. The first paper addressing split computing [KHG+17] concluded that, for certain models, splitting at the input or output layers minimized latency and energy consumption. This is equivalent to computing the whole inference locally or remotely. However, these methods heavily depend on the model architecture, and some recent models such as ResNet [HZRS16] lack early bottlenecks, which limits their effectiveness. Some studies have also used this approach to address privacy concerns by offloading only intermediate representations [JJLM18]. Others, like JointDNN [EAP21], extend this approach to include generative models and autoencoders for collaborative computation. In contrast to Neurosurgeon [KHG+17], they conclude that splitting results in optimal solutions in terms of latency and energy consumption.
- *With Bottleneck Injection:* These methods introduce artificial bottlenecks and modify the architecture to facilitate splitting. For example, encoding and decoding components can compress intermediate representations to reduce communication overhead [EEP19, MCB+20]. Approaches such as BottleNet [EEP19] and feature compression strategies [CB18] incorporate lightweight encoders to ensure efficient data transmission, even for models without natural bottlenecks. These methods have the disadvantage that they require some form of additional training after injection of the bottleneck.

Most studies [LLW+18, EAP21, PCMP21, KHG+17, LAB23, BVP+21, ZCX21, ZLL+20] evaluate various types of cost (e.g. computational load, energy consumption, communication cost) to partition models at each of their splitting points, while only a few choose splitting points based on heuristics [CB18, CCB20]. Some approaches make use of graph-based optimization, which models the DNN as a computational graph to analyze trade-offs [EAP21]. Emerging approaches, such as reinforcement learning, dynamically adjust split points based on system conditions [CLY+24]. These methodologies aim to balance computational load, communication overhead, and task-specific performance.

Although split computing effectively reduces latency and data transfer, many studies prioritize these metrics over energy consumption, which remains underexplored as a critical optimization target.

2.5.1 Motivational Scenario for Split Computing

Knowing the background and approaches in split computing, we can now consider a practical scenario that highlights the relevance of this technique. Figure 2.3 illustrates a

system in which users employ an edge device application to perform image classification tasks. This application uses a neural network (NN) and opportunistically engages cloud resources to perform inference operations.

During execution, the application decides whether to:

- perform inference entirely on the edge node (*edge-only*),
- fully delegate inference to a cloud node (*cloud-only*), or
- adopt a hybrid approach by splitting the NN into a *head model* processed on the edge and a *tail model* executed in the cloud.

This hybrid approach leverages the strengths of both edge and cloud computing. Edge-only inference minimizes latency by avoiding network overhead, but is constrained by the limited computational capacity of edge devices. Cloud-only inference, on the other hand, provides access to high computational resources but incurs additional latency due to data transfer. Split computing strikes a balance by processing some layers of the model locally and offloading the rest to the cloud, optimizing latency and resource utilization.

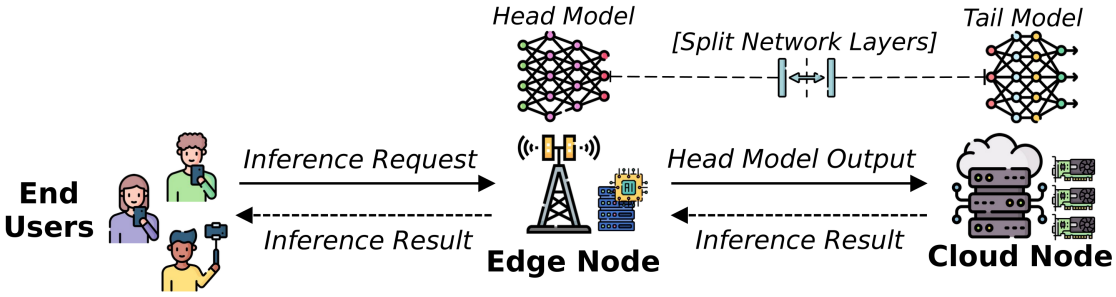


Figure 2.3: Motivational scenario for split computing: inference tasks can be executed on the edge, in the cloud, or through a hybrid approach that splits the computation between both [MTIB24].

This scenario demonstrates how split computing enables edge devices to perform complex tasks that would otherwise be infeasible, improving overall system performance. By balancing computation between edge and cloud, split computing supports the deployment of advanced AI models in real-world applications while addressing challenges related to latency, scalability, and resource constraints.

The next chapter reviews existing work on split computing and related approaches, exploring their methodologies and contributions to this field.

CHAPTER 3

Related Work

This chapter provides an overview of related work in energy-aware split computing, focusing on approaches without architectural modifications that align with our goals of easy deployment. First, we briefly discuss studies employing architectural modifications, emphasizing their strengths and limitations. Then, we analyze in detail energy-aware split computing works without architectural changes, highlighting similarities and differences with our approach. Finally, we summarize the current state of research and position our contributions.

3.1 Split Computing with Bottleneck Injection

Split computing approaches that introduce architectural modifications often aim to compress intermediate representations, improve communication efficiency, or tailor models for deployment in resource-constrained environments. While effective, these methods require significant retraining of neural networks, which limits their applicability for pre-trained models in real-world deployment. In the following, we summarize key studies in this category that are also energy-aware.

The *SPINN framework* [LVA+20] incorporates progressive inference by adding early exits to CNNs such as ResNet50 and MobileNetV2, allowing dynamic adjustments to accuracy, latency and energy based on input complexity. A multiobjective scheduler optimizes split points and early exit policies in real time. However, the need for extensive retraining to integrate early exits and optimize a specialized cost function reduces its practicality for deployment.

DeepCOD [YLL+20] leverages encoder-decoder structures for compressive offloading, targeting models such as ResNet50 and DeepSpeech. By integrating compressive sensing and knowledge distillation, it achieves high compression ratios (up to 1000x) with minimal

accuracy loss. However, this approach requires retraining the encoder-decoder pipeline, making it less suitable for deployment with pre-trained models.

BottleFit [MCS+22] replaces intermediate DNN layers with encoder-decoder bottlenecks to compress data for transmission. This method achieves up to 77.1% compression with a loss of accuracy of less than 0.6%. However, it requires multistage retraining, including pre-training and fine-tuning of the compressed layers, to maintain accuracy under strong compression.

SpikeBottleNet [HD24] applies spiking neural networks (SNNs) to split computing by introducing bottlenecks that transform intermediate representations into sparse spike events. This results in high compression ratios and significant energy savings, but requires retraining SNNs and compression modules, making it less applicable for pre-trained models.

These approaches demonstrate substantial energy improvements but depend on architectural modifications that require retraining. In contrast, our work focuses on split computing for pre-trained neural networks, avoiding the need for retraining. The next section examines studies that share this focus.

3.2 Split Computing without Architectural Modifications

This section reviews energy-aware split computing studies that operate on pre-trained models without requiring architectural changes. These approaches align closely with our focus on inference-only deployment scenarios. Table 3.1 summarizes key dimensions of these studies, including their tasks, datasets, neural network models, and performance metrics. We analyze the studies listed in detail, highlighting their methodologies, contributions, and limitations.

The foundational work of *Neurosurgeon* [KHG+17] introduced a system that dynamically partitions DNN computation between mobile devices and the cloud to optimize latency and energy consumption. It profiles each DNN layer to predict latency and energy under runtime conditions, such as network bandwidth and server load, and selects split points accordingly. Neurosurgeon's key strength lies in its lightweight runtime system, which minimizes computational overhead. However, its reliance on layer-specific regression models for performance prediction introduces inaccuracies, as the execution of consecutive layers does not always follow a linear trend. In contrast, `DynaSplit` uses application-specific profiling for greater precision, capturing the actual performance of split points under the given application configuration. Neurosurgeon also focuses solely on layer partitioning and does not address hardware-level optimizations, such as tuning CPU or TPU frequencies, which `DynaSplit` incorporates to further improve energy and latency trade-offs.

JointDNN [EAP21] models split computing as a shortest-path optimization problem in a directed acyclic graph (DAG). The nodes represent DNN layers, and the edges encode computation and communication costs. Using integer linear programming (ILP),

JointDNN dynamically determines optimal split points to minimize latency or energy consumption while adhering to specific constraints, such as battery life or QoS. Unlike Neurosurgeon, JointDNN extends its applicability to generative models and autoencoders. However, JointDNN does not include hardware tuning as part of its optimization strategy, which limits its ability to further enhance efficiency, which contrasts with our work `DynaSplit`.

CRIME [PCMP21] focuses on optimizing RNN inference across a collaborative network of devices. Rather than splitting computations across multiple devices, CRIME assigns entire inference tasks to a single device, dynamically selecting the optimal device based on local estimations of execution and transmission times. This decentralized decision-making allows CRIME to adapt effectively to changing network conditions, workloads, and input sizes without requiring global coordination. However, CRIME is specialized for RNNs and does not generalize well to other DNN architectures. In contrast, `DynaSplit` 's profiling and optimization framework supports a wide range of architectures, including transformers. Furthermore, `DynaSplit` operates in a fixed edge-cloud setup, whereas CRIME focuses on distributed device networks, making it less applicable to scenarios requiring centralized coordination.

Labriji et al. [LMAS23] propose a dynamic resource allocation algorithm for energy-efficient cooperative inference. This method uses Lyapunov stochastic optimization to select split points and communication resources based on real-time observations of system parameters, such as wireless channel conditions and edge server CPU availability. Unlike `DynaSplit`, which assumes a fixed edge-cloud pair, Labriji et al. include the selection of the offloading device as part of their optimization. This broader scope allows them to address scenarios with multiple offload options, although it makes their approach less applicable to fixed deployment setups such as `DynaSplit` 's. Additionally, their algorithm does not consider hardware parameter tuning, which limits its ability to explore energy-latency trade-offs at a finer granularity.

Xiao et al. [XXW+23] employ a reinforcement learning framework to optimize split points and edge server selection for collaborative inference. Their approach dynamically adjusts the partitioning of DNNs while simultaneously choosing the most suitable edge server to balance latency and energy consumption. This dual optimization scope distinguishes Xiao et al. from `DynaSplit`, which operates in a fixed edge-cloud setup. Although effective, the focus on device-level energy consumption excludes cloud energy considerations, which `DynaSplit` incorporates for a more holistic optimization. Both studies have in common that they consider the energy of all devices involved in the split computation.

Yang et al. [YW23] adopt a coalition formation game to optimize user associations and resource allocation in privacy-sensitive split inference for edge camera networks. Their framework integrates privacy, energy efficiency, and detection accuracy into a unified utility function, encouraging overlapping-view cameras to offload tasks to the same counterpart for enhanced accuracy. Unlike `DynaSplit`, which focuses on energy and latency trade-offs, Yang et al. explicitly optimize for privacy and detection accuracy, making their approach well suited for specialized applications in edge camera networks.

However, their focus on privacy and overlapping camera views makes their method less applicable to more general-purpose edge-cloud setups.

Liu et al. [LFEF24] introduce a two-split DNN inference strategy in a three-tier architecture consisting of an edge device, a near-edge accelerator, and the cloud. Their approach uses Pareto front optimization to balance latency and energy, introducing autoencoders to compress intermediate feature maps and reduce communication overhead. Although this method achieves fine-grained trade-offs, the reliance on autoencoder training introduces more overhead during deployment. `DynaSplit` avoids any training entirely and focuses on single-split configurations, making it more practical for deployment in simpler edge-cloud setups.

Zhao et al. [ZSL+24] present a two-timescale optimization framework for UAV-based split inference. Their approach integrates Tiny Reinforcement Learning for split point selection and Optimization Programming for resource allocation, targeting energy efficiency and delay constraints for sequential tasks. Although effective for UAV navigation, the method of Zhao et al. is highly specialized and lacks generalizability to various edge-cloud setups. In contrast, `DynaSplit` provides a more general framework suitable for a wide range of edge-cloud applications.

Li et al. [LB24] focus on energy-efficient split inference in wireless sensing systems with multiple users. Their framework integrates deep-reinforcement learning to determine optimal split points and convex optimization for resource allocation, addressing the unique challenges of multiuser setups such as indoor crowd counting and action recognition. Although their approach is effective for this specialized use case, it does not generalize to broader edge-cloud deployments. `DynaSplit`, on the other hand, targets a fixed edge-cloud pair, allowing a wider applicability to generic inference scenarios.

Zhang et al. [ZZL+24] propose DVFO, which integrates dynamic voltage and frequency scaling (DVFS) with reinforcement learning to optimize energy and latency for edge devices. Although `DynaSplit` shares the concept of hardware tuning, `DynaSplit` extends this to include also the hardware parameter of cloud devices and additionally considers cloud energy in optimization, providing a more comprehensive view of energy consumption.

3.3 Summary

The reviewed studies highlight diverse approaches to energy-aware split computing, focusing on dynamic partitioning, collaborative resource management, and domain-specific optimizations. Although these methods effectively address latency, energy, and accuracy trade-offs, they often lack considerations such as total system energy, comprehensive hardware tuning, or support for transformer architectures. In addition, many approaches are tailored to specific applications, limiting their generalizability to broader edge-cloud setups.

This work introduces DynaSplit, a framework that addresses these gaps through the following contributions:

- DynaSplit employs application-specific profiling for accurate performance measurements, overcoming the limitations of prediction-based approaches such as Neurosurgeon.
- It incorporates hardware parameter tuning for both edge and cloud devices, enabling finer control over energy-latency trade-offs.
- By assuming a fixed edge-cloud pair, DynaSplit simplifies deployment compared to methods like Labriji et al. and Xiao et al., which include offloading device selection as part of the optimization.
- DynaSplit explicitly supports transformer architectures, which addresses a gap in existing energy-aware split computing studies.
- Unlike most previous work, DynaSplit considers the total energy consumption of all devices involved in the inference process, extending the scope of optimization beyond edge-only setups.

By integrating these features into a unified hardware-software co-design framework, DynaSplit provides a scalable and efficient solution for energy-aware, latency-sensitive inference across diverse edge-cloud deployments, advancing the state of the art in split computing.

Work	Task	Dataset	Models	Metrics
Kang et al. [KHG+17]	Image classification Speech recognition NLP	N/A	AlexNet VGG19 DeepFace LeNet-5 Kaldi SENNA	**D, E, L**
Eshratifar et al. [EAP21]	Image classification Speech recognition	N/A	AlexNet OverFeat NiN VGG16 ResNet50	**D, E, L**
Pagliari et al. [PCMP21]	Time-series analysis NLP	SNLI SQuAD IMDB	RNNs	**E, L**
Labriji et al. [LMAS23]	Image classification	ImageNet	MobileNetV2	**E, L**
Xiao et al. [XXW+23]	Object detection	ImageNet	AlexNet VGG19	**E*, L**
Yang et al. [YW23]	Object detection	Caltech101	VGG19	**A, E, P**
Liu et al. [LFEF24]	Image classification	CIFAR-100 ImageNet	VGG16 ResNet50	**A, E, L**
Zhao et al. [ZSL+24]	Image classification	N/A	Custom CNN	**E, L**
Li et al. [LB24]	Wireless sensing	N/A	Custom CNN	**E, L**
Zhang et al. [ZZL+24]	Image classification Speech recognition	CIFAR-100 ImageNet	EfficientNet-B0 ViT-B16	**A, E, L**
This work	Image classification	ImageNet	VGG16 ResNet50 MobileNetV2 ViT-B16	**A, E*, L**

Table 3.1: Overview of prior work and comparison to this study analogously to [MLR23]. Metrics: **A**: Model accuracy, **D**: Transferred data size, **E(*)**: Energy consumption (of all devices), **L**: Latency, **P**: Privacy.

CHAPTER 4

Definitions and Problem Formulation

In this chapter, we provide detailed definitions that are essential for the reader's comprehension and proceed to define and elaborate on the associated optimization problem.

4.1 Notation

We utilize the following notation to differentiate between specific moments and durations:

- **Points in time** are indicated by lowercase letters like t_0, t_{net1}, and t_{net2}, each specifying a distinct point during the inference phase.
- **Time spans** (durations or intervals) use uppercase letters such as T_{edge}, T_{net}, and T_{cloud}, representing the length of time allocated for particular tasks.

4.2 Model Partitioning

Consider a neural network (NN) model M, consisting of L layers, which is divided into two segments: a head segment indicated by M_h, containing the first k layers, and a tail segment denoted by M_t, consisting of subsequent $L - k$ layers. The segment M_h is executed on the edge device, while the segment M_t is handled by the cloud server. After processing the head segment M_h, the edge device transmits the intermediate results to the cloud server, allowing further processing by the tail segment M_t. The selection of the split layer k can be adjusted within a range of 0 to L, thus covering a variety of specific configurations:

- $k = 0$: Here, the model is run entirely in the cloud, with the edge device simply transmitting input data.
- $k = L$: In this case, the model runs entirely on the edge device, eliminating the need for data transfer to the cloud.

Changing k, it is possible to dynamically distribute the neural network inference task between the edge and the cloud. This approach provides crucial adaptability to balance objectives such as latency, energy consumption, and accuracy, while respecting model limitations and application QoS requirements. Edge devices typically have limited computational capacity relative to cloud servers, affecting their ability to handle many latency-sensitive requests. In contrast, cloud systems offer scalable resources and predominantly consume energy during active processing, as opposed to data transfer or idle times. Modern cloud models, particularly serverless computing, facilitate these on-demand computing services [LZY+22].

4.3 Configuration Space

The latency and energy consumption during inference are significantly impacted by the choice of split layer and various hardware configurations. Modifying computing unit frequencies and leveraging accelerators can greatly affect this process. These elements are all components of a system configuration. Our objective is to identify the optimal configuration settings for a specified inference task. The configuration space X includes suitable hardware and software parameters, including the split layer, CPU frequency adjustments at the edge, the use of edge hardware accelerators, and the deployment of cloud GPU resources.

4.4 Latency Model

The cumulative inference duration $T_{\text{inf}}(x)$ for a given setup $x \in X$ consists of these separate times:

- **Edge Inference Time $T_{\mathbf{edge}}(x)$:** Time consumed by executing the initial part M_h on an edge device.
- **Network Transfer Time $T_{\mathbf{net}}(x)$:** Duration required for data transmission between the edge and the cloud, covering both the delivery of intermediate results and the reception of final output.
- **Cloud Inference Time $T_{\mathbf{cloud}}(x)$:** Time spent processing the latter segment M_t on a cloud platform.

Thus, the total inference duration is expressed as:

$$T_{\text{inf}}(x) = T_{\text{edge}}(x) + T_{\text{net}}(x) + T_{\text{cloud}}(x)$$

considering these distinct cases:

- For $k = 0$ (Cloud-Only Inference): $T_{\text{edge}}(x)$ is negligible since the full computation is managed by the cloud, yet minimal tasks such as data preparation occur on the edge device.
- For $k = L$ (Edge-Only Inference): Both $T_{\text{cloud}}(x) = 0$ and $T_{\text{net}}(x) = 0$, as every calculation is completed locally on the edge device, eliminating the need for cloud data transfer.

The network transfer time $T_{\text{net}}(x)$ becomes significant only when $k < L$, indicating the inference is shared between the edge and the cloud.

4.5 Energy Model

The total energy consumption, denoted as $E_{\text{inf}}(x)$, comprises the energy consumed by both edge and cloud devices. The energy is determined by integrating the power over specific time intervals. For the edge device, this involves calculating the energy throughout the inference time frame, starting at t_0 and ending at t_{inf}, which is equivalent to the complete inference period T_{inf}. In contrast, for the cloud device, the energy is calculated only during its active processing period, beginning at t_{net1} and ending at t_{net2}, indicating the initiation and termination of the cloud computation phase. Although we consider the energy utilization by both edge and cloud devices during their computation periods, we do not include the energy used by network infrastructure components, such as routers or switches, during data transmission. This omission is due to the practical challenges of measuring network energy consumption [SGSB+15] and its relatively insignificant effect.

As a result, the overall energy consumption is represented by:

$$E_{\text{inf}}(x) = \begin{cases} \int_{t_0}^{t_{\text{inf}}} P_{\text{edge}}(t, x)\, dt & \text{if edge-only computation,} \\ \int_{t_0}^{t_{\text{inf}}} P_{\text{edge}}(t, x)\, dt + \int_{t_{\text{net1}}}^{t_{\text{net2}}} P_{\text{cloud}}(t, x)\, dt & \text{otherwise.} \end{cases}$$

where:

- t_0 denotes the start of the entire inference procedure, which can involve beginning the edge inference or organizing data for transfer to the cloud.
- t_{net1} represents the moment when the transfer of data from the edge to the cloud ends and the cloud is set to start processing. This stage is relevant only if computation is involved in the cloud (that is, $k < L$).
- t_{net2} is the point at which the cloud completes the processing and returns the results to the edge. Similarly to t_{net1}, this stage applies only if the cloud computation occurs ($k < L$).

- t_{inf} marks the conclusion of the inference procedure, either when the edge computation is complete (for edge-only scenarios) or when the edge obtains the results from the cloud.

Figure 4.1 shows an example of how total energy is computed over time using the notation introduced above.

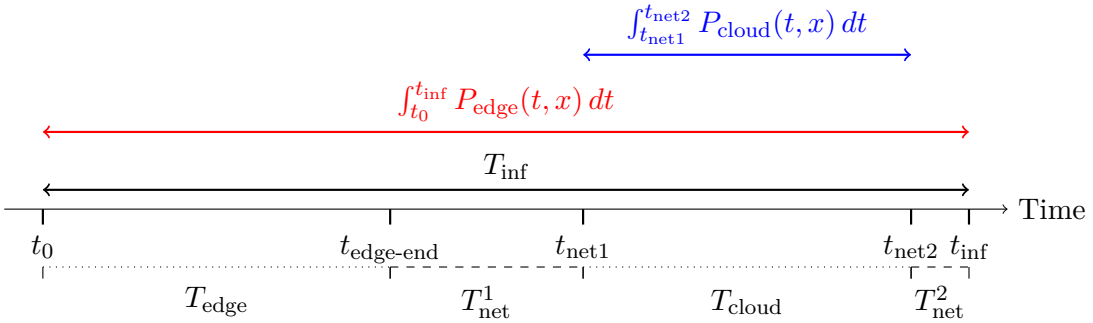


Figure 4.1: Energy consumption and time intervals throughout the inference process. The timeline illustrates edge computation (T_{edge}), data transfer to the cloud (T^1_{net}), cloud computation (T_{cloud}), and the return transfer to the edge (T^2_{net}). Edge energy is measured across the entire duration of inference, whereas cloud energy is only accounted for during the cloud computation phase.

4.6 Optimization Problem

`DynaSplit` primarily seeks to minimize latency and energy consumption while simultaneously increasing the accuracy of inference through a *hardware-software co-design framework*.

This is organized in the form of a multi-objective optimization problem (MOOP) and can be depicted as follows:

$$\text{minimize}_{x \in X} \left(T_{\text{inf}}(x), E_{\text{inf}}(x), -A(x)\right) \tag{4.1}$$

where $A(x)$ denotes the accuracy of inference for a particular configuration x, serving as a function that maps from the configuration space X to the corresponding accuracy value.

Finding solutions to the MOOP generates a set called *Pareto front*. This set comprises *non-dominated configurations*, which means that no alternative configuration in the search space can improve one objective without compromising at least one other [NZES05]. Essentially, the Pareto front highlights the best trade-offs among objectives by offering a spectrum of configurations that balance these competing criteria. For example, obtaining lower latency might lead to higher energy consumption, but still deliver superior accuracy.

A configuration $x^* \in X$ is considered non-dominated if there is no configuration $x \in X$ such that:

$$f_i(x) \leq f_i(x^*) \quad \text{for every } i = 1, 2, \ldots, m$$

and

$$f_j(x) < f_j(x^*) \quad \text{for at least one } j.$$

Here, $f_i(x)$ represents the i-th objective, with m being the total number of objectives. The set of all these non-dominated configurations constitutes the Pareto front $\mathcal{P}$:

$$\mathcal{P} = \{x^* \in X \mid \text{there is no } x \in X \text{ that dominates } x^*\}.$$

In our scenario, the goals are to minimize the total inference latency $T_{\text{inf}}(x)$, minimize the total energy consumption $E_{\text{inf}}(x)$, and maximize the accuracy $A(x)$.

CHAPTER 5

DynaSplit: Methodology and Solution Approach

This chapter outlines DynaSplit's methodology and solution strategy. It starts with highlighting crucial observations on how hardware and software setups affect system performance and define the search space for exploration. The parts of the core system are described, followed by details of DynaSplit's *Offline Phase* and *Online Phase*.

5.1 Preliminary Observations

We analyzed a variety of neural networks in split computing settings by adjusting various hardware and software configurations to gain deeper insights into the previously mentioned scenario through empirical data. To facilitate this, we established an edge-cloud testing environment that included a Raspberry Pi 4B (featuring a quad-core ARMv8 processor and 8 GiB of RAM), supported by a Google Coral TPU serving as an edge accelerator, alongside a cloud node (featuring dual Intel Xeon E5-2698 v4 CPUs and 512 GiB of RAM) equipped with 8 NVIDIA Tesla V100 GPUs (although only a single GPU was used for the tests). Using the ImageNet dataset [RDS+15], we evaluated four pre-trained neural networks. which consisted of ResNet50 [HZRS16] containing 0.85 million parameters, MobileNetV2 [SHZ+18], VGG16 [SZ15] with 138 million parameters and Vision Transformer (ViT) [DBK+21] housing 86 million parameters. Collectively, we generated 1,000 inference requests by randomly choosing images from the ImageNet validation dataset, simulating a user workload. In addition, we collected data on latency, energy expenditure, and accuracy to enhance our analysis.

Throughout our preliminary experiments, it became evident that split computing is remarkably advantageous for neural networks possessing a vast number of parameters, such as VGG16 and Vision Transformer. In contrast, smaller models such as ResNet50

and MobileNetV2 did not gain any advantage from split computing. These smaller models perform with greater speed and lower energy usage when using edge-based deployments only. On the other hand, VGG16 and ViT exhibited significant latency improvements by using both edge and cloud resources. This outcome underscores the potential of split computing for larger networks. Our motivation is driven by the intention to examine runtime configurations that effectively employ the capabilities of both edge and cloud systems. This approach aims to improve the performance of computation-heavy models while decreasing the total energy consumption. Given these insights, VGG16 and ViT have been selected as the networks of interest for our investigation.

In addition, we employ the VGG16 model to thoroughly investigate how different parameters influence inference tasks. VGG16 represents a convolutional neural network (CNN) with up to 22 layers, as detailed in its open-source Keras implementation. We identify the following possible runtime configurations:

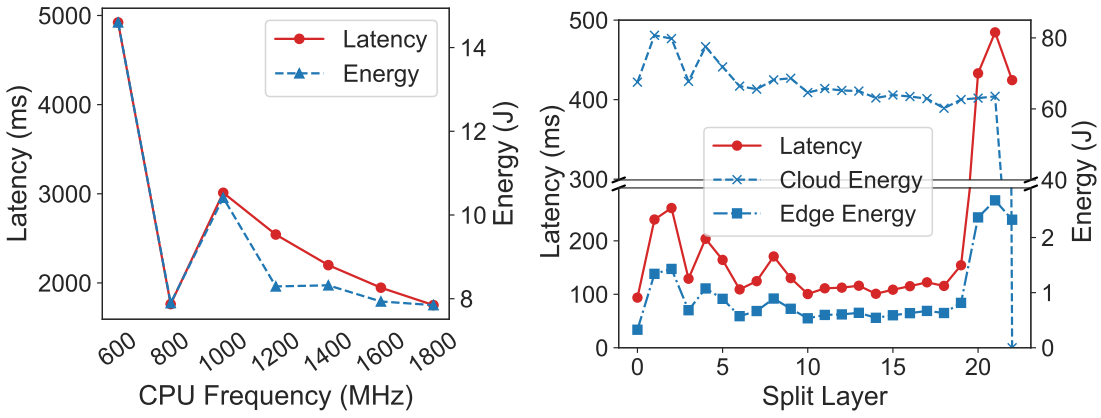
1. Modifying the CPU frequency on the edge node.
2. Choosing the use of a hardware accelerator, such as a TPU, on the edge node or a GPU on the cloud node.
3. Segmenting the CNN architecture into a dynamic head model and tail model, enabling the execution of specific layers on the edge node while the remaining layers run on a cloud node.

In Figure 5.1, Figure 5.2, and Figure 5.3 the influence of particular configuration settings is illustrated on three different metrics: the mean latency measured in milliseconds, the mean energy usage quantified in Joules, and the mean accuracy of inference queries.

In Figure 5.1a, we present the results of running exclusively on the edge by adjusting the CPU frequency of the Raspberry Pi 4B, intentionally excluding the TPU edge accelerator. Thus, we can investigate the effect of CPU frequency on the inference process. The data indicates that elevating the CPU frequency results in reduced latency and energy consumption. Initially, both metrics drop notably, although the rate of decrease slows as the frequency is further increased. Remarkably, we detected anomalies at the 800 MHz setting across repeated trials, which deviate from the general pattern.

Figure 5.1b illustrates the impact of the model split layer on both latency and energy usage. To illustrate, when division occurs at layer 5, the initial five layers are processed on the edge node, while the subsequent layers are executed on the cloud node. In this case, the TPU is employed as the edge accelerator, operating at its maximum frequency of 500 MHz, and the CPU runs at 1,800 MHz. Furthermore, the cloud node uses a GPU for acceleration. The data show that inference is the fastest when done solely on the cloud device. Also, we can see that edge computation takes a significantly longer duration. The split solutions between show non-linear behavior. The energy consumption follows a pattern similar to the latency. However, finding an optimal split point proves

to be complex as the relationship between the split point and both latency and energy consumption is intricate.



(a) (b)

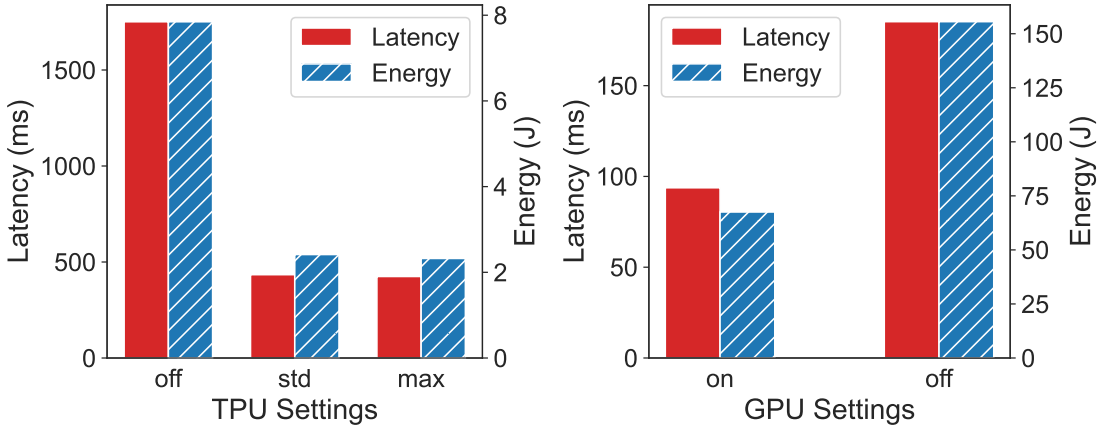
Figure 5.1: This figure illustrates the effects of adjusting CPU frequency and modifying the split layer on the latency experienced and energy consumed during inference tasks when utilizing the VGG16 network. Results are derived from averaging data collected over 1,000 inference operations [MTIB24].

In our analysis of edge acceleration within edge-only configurations, the TPU was either inactive, running at its standard 250 MHz frequency, or at its maximum 500 MHz frequency. Figure 5.2a illustrates how these settings affect energy consumption and latency. While the TPU consumes additional power, it delivers $\sim 3\times$ lower average energy usage than CPU-based computation, due to accelerated processing and shorter inference times. Interestingly, there is no substantial discrepancy in performance between the 250 MHz (std) and 500 MHz (max) TPU settings for this network.

Furthermore, we explored the impact of cloud acceleration with and without GPU in cloud-only scenarios, as depicted in Figure 5.2b. The results reveal that GPU acceleration decreases both latency and energy usage in the cloud environment by about a third.

In our last preliminary analysis, we assess how the use of edge acceleration and splitting layers influences the accuracy of the inference. The accuracy is expected to generally decline as more model layers are processed on an edge node, due to the necessary quantization of the model into 8-bit integers to enable operation on edge devices with limited resources [DLH+20, CMGS20]. Surprisingly, our findings indicate that any changes in accuracy remain minimal, consistently falling within a sub-percent margin, as illustrated in Figure 5.3. Although there is a slight reduction in accuracy when the final layers are executed on the edge, this effect does not distinctly favor either the TPU or CPU, with the minor discrepancies likely attributed to numerical variations.

We have found that latency and energy consumption during inference tasks are significantly impacted by a mix of hardware-software parameters, with minimal effect on accuracy,



(a) (b)

Figure 5.2: Influence of utilizing TPU settings, including adjustment of frequency, alongside GPU settings on both the inference duration and energy consumption for the VGG16 architecture [MTIB24].

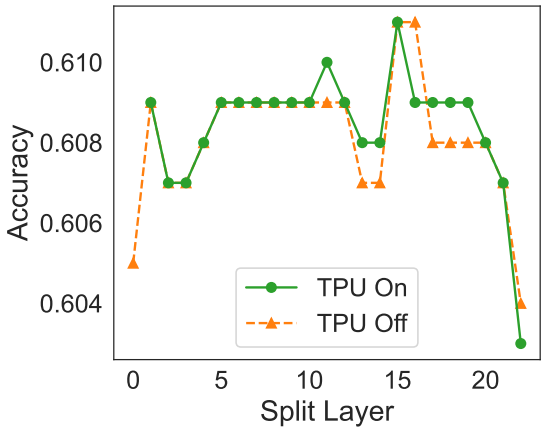


Figure 5.3: This visualization illustrates how adjusting the split layer and using TPU resources can influence the accuracy outcomes for the VGG16 architecture [MTIB24].

which leads us to the configuration space depicted in Table 5.1. This underscores the importance of developing sophisticated runtime configuration and scheduling methods for neural network-based inference in edge-cloud frameworks. These systems must adhere to quality of service standards while striving for energy efficiency.

5.2 System Overview

We present `DynaSplit`, a framework that prioritizes energy efficiency by combining software parameters (e.g., the neural network split layer) with hardware parameters (such

Table 5.1: Classification and extent of various hardware and software parameters [MTIB24].

Parameter	Parameter Type	Domain
CPU Frequency (CPU_f)	Numerical (low: 0.6, high: 1.8, step: 0.2)	{0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8}
Edge TPU Frequency (TPU_f)	Categorical	{off, std, max}
Use Cloud GPU (GPU)	Categorical	{Yes, No}
VGG16 [SZ15] Split Layer (L_{VGG})	Numerical (low: 0, high: 22, step: 1)	{0, 1, 2, ..., 22}
Vision Transformer [DBK+21] Split Layer (L_{ViT})	Numerical (low: 0, high: 19, step: 1)	{0, 1, 2, ..., 19}

as accelerator use and CPU frequency) in a *co-design optimization framework*. This framework aims to efficiently handle and respond to inference requests at the edge nodes.

Using `DynaSplit`, an in-depth investigation of extensive parameter search spaces is performed to uncover the optimal execution settings. This facilitates dynamic tuning of both edge and cloud computing nodes and supports strategic scheduling of inference tasks.

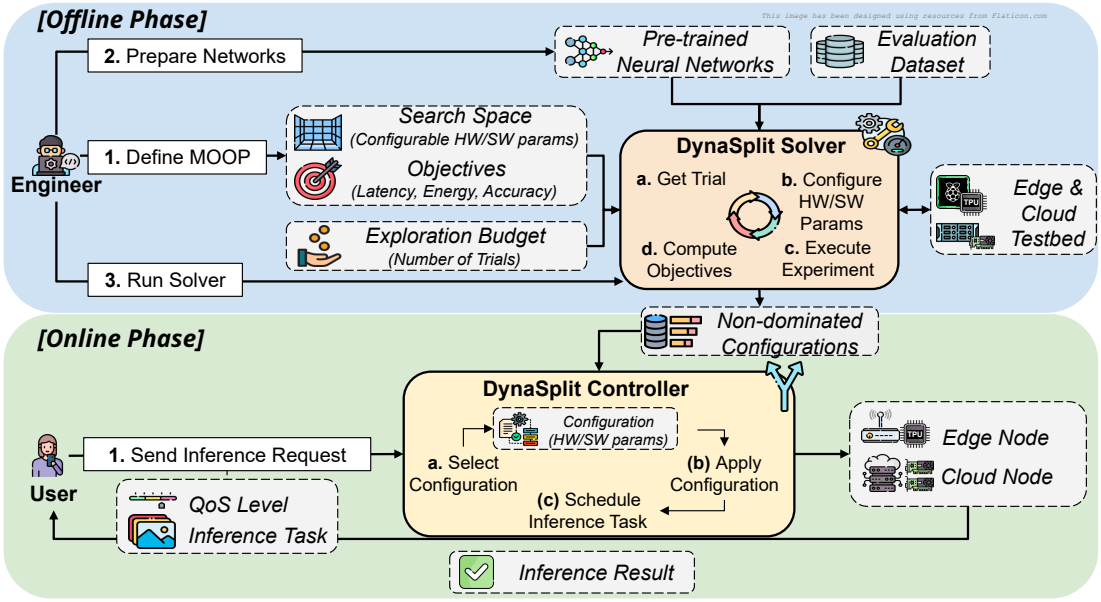


Figure 5.4: This diagram offers an extensive illustration of the `DynaSplit` framework, detailing its structure and components [MTIB24].

In Figure 5.4, a comprehensive outline of the `DynaSplit` framework is depicted, highlighting its two essential components: *`DynaSplit` Solver* and *`DynaSplit` Controller*. These integral parts coordinate the separate phases of the framework's process, identified as *Offline Phase* and *Online Phase*.

The *DynaSplit Solver* tackles the multi-objective optimization problem by thoroughly investigating the hardware-software search domain. Using a metaheuristic approach, it identifies configurations that are near optimal, which are subsequently used during execution to determine model partitioning points and allocate tasks to edge and cloud nodes. This methodology proves beneficial in the navigation of an expansive search domain, especially when the computational expense of assessing optimization goals is extremely high [TMI+23].

The *DynaSplit Controller* is responsible for organizing the schedule of incoming inference requests, focusing mainly on choosing the most energy-efficient setup identified by the *DynaSplit Solver* during the *Offline Phase*. This setup must meet the required quality of service requirements. The method for selecting the configuration guarantees that the specified inference latency limit is adhered to, consuming minimal energy and maintaining the inference task's precision.

5.3 Offline Phase

Within the *Offline Phase*, there are three main activities. Initially, the multi-objective optimization problem needs to be established by specifying the optimization goals and identifying the relevant hardware and software characteristics that form the search space. Subsequently, it is crucial to have pre-trained neural networks ready to handle any possible combinations of head and tail models. Lastly, the engineer can execute the *DynaSplit Solver*, enabling exploration of the search space to uncover the configurations that are not dominated.

5.3.1 Defining the MOOP

In accordance with section 4.6, DynaSplit aims to optimize three main goals: reducing energy consumption, decreasing inference latency, and improving inference accuracy. However, DynaSplit's scope is not limited to these objectives. The search space X, made up of both hardware and software parameters, is structured as a collection of configuration tuples. Specifically, X encompasses four distinct parameters: the CPU frequency at the edge node (CPU_f), the frequency of the edge hardware accelerator (TPU_f), the use of a cloud-based GPU, and the neural network split layer ($L_{\{network\}}$). Each parameter domain varies in cardinality, as detailed in Table 5.1. Consequently, the size of X is calculated as $|X| = |CPU_f| \times |TPU_f| \times |GPU| \times |L_{\{network\}}|$ configuration tuples. For example, for the VGG16 network, the equation yields $|X| = 7 \times 3 \times 2 \times 23 = 966$.

Initially, the search space X includes every possible combination of parameters. However, certain parameter configurations are not feasible due to the conditional structure of our search domain. Specifically, the assignment of a particular value v to the parameter p_1 depends on the value w that is assigned to the parameter p_2. In detail,

- *TPU utilization in exclusively cloud-based inference ($k = 0$):* When computations

are entirely executed on the cloud node and bypass edge inference, the TPU remains unused.

- *GPU utilization in exclusive edge-based inference ($k = L$):* When computations occur entirely at the edge node, avoiding cloud processing, GPU is not utilized.

Moreover, every neural network might impose additional restrictions arising from its architectural and hardware restrictions, prompting the need to specify a search space for each network individually. Two pre-trained NNs were evaluated in our study, specifically VGG16 [SZ15] and Vision Transformer (ViT) [DBK+21]. The VGG16 model, a convolutional neural network (CNN), has been extensively applied in the field of image recognition in the last ten years [ARAD21, KG19, CZ20]. In contrast, ViT represents a transformer-based neural network, which is rapidly gaining popularity within the research sector due to its exceptional performance in vision-centric tasks [KNH+22]. With specific regard to ViT, implementation using edge TPU is omitted in all configurations due to specific memory restrictions [WZWY22], rendering scenarios with the TPU_f parameter set at 250 or 500 infeasible.

By eliminating VGG16 configurations that are not feasible, we narrow the search space to a total of 917 viable configurations, effectively omitting 5.07% of the initial search space. Likewise, for ViT, the number of feasible configurations is refined to 273, thus excluding 2.50%. This strategic reduction in the search space not only significantly reduces the computational workload required but also maintains the integrity of the solution quality, thus enabling us to concentrate on the most promising configurations available.

5.3.2 Preparing the Networks

The model partitioning is tailored to suit the unique architecture of each neural network, ensuring segmentation only occurs where information can flow continuously [MLR23]. As a result, the number of layers available for division varies between networks. For example, within the VGG16 network, segmentation is feasible after each separate layer, totaling 22 opportunities for partitioning. In contrast, the ViT network can be divided at the level of its complete transformer encoder blocks, totaling 19, due to its inherent structural dependence.

Moreover, it is essential for the network layers to be prepared for deployment on specialized hardware, such as an edge TPU. Specifically, the head sections of the network must undergo quantization into 8-bit integers and be compiled exclusively for TPU deployment. This method, termed post-training quantization, can result in minor changes in accuracy [CMGS20, DLH+20]. We address these potential accuracy fluctuations by incorporating accuracy as one of the key optimization criteria in our optimization strategy.

For certain networks, size constraints can hinder the ability to perform quantization and execute them on an edge hardware accelerator, as for ViT [WZWY22]. Therefore, in such scenarios, the initial segments can be processed using conventional 32-bit floating point operations on the edge node, relying on computations handled by the CPU.

5.3.3 Running the `DynaSplit` Solver

To discover the Pareto front solutions, also known as non-dominated configurations (these are the configurations that meet the optimization goals to diverse but notable extents), it is essential to compute the multi-objective function specified in Equation 4.1.

To address the MOOP, `DynaSplit` utilizes NSGA-III [DJ14, JD14], an efficient and effective multi-objective optimization algorithm recognized for its effectiveness in a variety of fields [Fie17, MPCD19, FDHM22]. Specifically tailored for scenarios with multiple objectives, such as minimizing latency and energy use while maximizing accuracy, non-dominated Sorting Genetic Algorithm (NSGA)-III enhances its predecessor, NSGA-II, by incorporating a reference point-based methodology. This advancement ensures a well-distributed array of solutions throughout the objective space.

The fundamental process of NSGA-III consists of forming a set of candidate solutions and systematically improving these solutions through iterative evaluations for a variety of objectives. Unlike algorithms that focus on a single objective, NSGA-III categorizes solutions into various nondomination levels, where each rank includes solutions that are not outperformed by any other solution. This method enables the algorithm to concentrate on preserving the variation within the *Pareto-optimal solutions*.

The NSGA-III algorithm utilizes a set of predetermined reference points to direct its search operations, which guarantees a uniform distribution of solutions along the Pareto front. This characteristic is especially critical in scenarios involving the optimization of more than two objectives, as it deters the algorithm from concentrating solutions in limited sections of the objective space. This methodology enables NSGA-III to effectively manage and balance the compromises between various factors such as energy efficiency, latency, and accuracy.

To minimize computational demands, we deploy NSGA-III on only *20% of the available search space*, allowing it to still effectively approximate the *Pareto front*. Concentrating on this fraction of the search space reduces computational requirements, yet still extracts a meaningful collection of superior solutions for use during *Online Phase*. This selective approach, when combined with the reference point scheme, empowers NSGA-III to explore an array of diverse, optimal settings that harmonize energy consumption, latency, and accuracy.

The strength of NSGA-III in comparison to NSGA-II is its capacity to manage problems involving three or more objectives while maintaining the diversity of the solution space. This capability is especially important for `DynaSplit`, which demands simultaneous optimization of energy, latency, and accuracy across multiple edge-cloud configurations. The method generates a Pareto front approximation that promotes solution diversity and provides numerous high-quality trade-offs. Consequently, `DynaSplit` becomes both scalable and adaptable to varying workloads and hardware environments.

For every proposed configuration under examination, called a trial, the *`DynaSplit` Solver* adapts the edge-cloud testbed to align with this configuration. This adjustment

involves loading the head and tail networks and configuring all hardware settings listed in the search space to match the distinct values of the configuration. Subsequently, the inference task is performed utilizing samples drawn from the evaluation data set. Throughout the experiment's execution, data concerning the objective values, such as inference latency, energy consumption during inference, and accuracy, are gathered and archived. Subsequently, these objective values are analyzed by the optimization algorithm to assess the effectiveness of the solution.

Throughout its process, the *DynaSplit Solver* archives every objective value that it assesses. Once the operation concludes, it retrieves the subset of configurations that are non-dominated from the entire result dataset. According to our empirical study, investigating only 20% of the search space is adequate to pinpoint effective configurations, compared to a broader search of approximately 80%. This finding is crucial because determining how each individual configuration meets the objectives requires collecting a multitude of empirical data points.

To ensure reliable performance metrics, DynaSplit conducts an average of 1,000 inferences per configuration. This approach accounts for testbed variability and gathers a sufficient sample size for metrics with a slower sampling rate compared to the individual inference duration, primarily because of physical constraints like power meters. For example, evaluating the 917 valid configurations of the VGG16 network requires *917,000 inferences*, which requires substantial processing resources and time. This underscores both the complexity and the computational expense involved in the optimization challenge addressed by DynaSplit. The ability to efficiently explore the configuration space thus substantially diminishes both the computational burden and the duration required for optimization.

5.4 Online Phase

During the *Online Phase*, the *DynaSplit Controller* plays a crucial role by handling the user's request, which includes both the inference task, like classification of an array of images, and the desired quality of service (QoS) level, defined by the maximum acceptable inference latency in milliseconds. The determination of QoS levels can arise from various sources such as service-level agreements (SLAs), application types (for example, distinguishing between time-sensitive and non-time-sensitive applications), or the nature of the mobile network connection being used (for instance, LTE, 4G, or 5G). Upon receiving the request, the *DynaSplit Controller* undertakes three vital operations, which are as follows:

1. Identify a configuration that is not only energy efficient, but also capable of satisfying the specified QoS requirement.

2. Apply this configuration by adjusting software and hardware settings across both edge and cloud infrastructures.

3. Execute the inference by effectively scheduling the requested task to ensure successful execution.

5.4.1 Selecting Configuration

The *DynaSplit Controller* leverages non-dominated configurations discovered in the *Offline Phase* to choose the optimal setup for a new request. Upon receiving an inference request, the user must specify a QoS requirement, usually denoting the highest permissible inference time, or latency. At system initialization, these non-dominated configurations are sorted and stored in memory based on two criteria:

1. Sorted by energy consumption in ascending order (lower is preferred).

2. Sorted by accuracy in descending order (higher is preferred).

Algorithm 5.1 Procedure for Scheduling and Configuring Requests [MTIB24].

Require: qos, the QoS level expressed as maximum inference latency (ms)
Require: $sortedConfigSet$, the sorted non-dominated configuration set
1: $config \leftarrow sortedConfigSet[0]$
2: **for** $i \leftarrow 0, \text{size}(sortedConfigSet)$ **do**
3: **if** $sortedConfigSet[i]$.latency $\leq qos$ **then**
4: **return** $sortedConfigSet[i]$
5: **end if**
6: **if** $sortedConfigSet[i]$.latency $< config$.latency **then**
7: $config \leftarrow sortedConfigSet[i]$
8: **end if**
9: **end for**
10: **return** $config$

In Algorithm 5.1, we present the pseudo-code for our configuration selection approach. The process starts by picking the first configuration in the ordered set (line 1), indicating the one with the highest energy efficiency due to the sorting. Subsequently, the search proceeds to identify the *most energy-efficient configuration that meets the necessary QoS threshold*; specifically, a configuration whose inference duration does not exceed the QoS requirement. This configuration is chosen if it is discovered (lines 2 - 4). If not, `DynaSplit` advances by seeking the *fastest configuration on hand*, even if it exceeds the QoS requirement (lines 6 - 10). This guarantees that the system optimizes performance and minimizes possible quality of service violations. This strategy allows `DynaSplit` to rapidly make decisions, which is vital for applications with low latency demands, such as autonomous vehicle control and real-time video processing. The algorithm's runtime complexity is $O(n)$, with n being the count of non-dominated configurations within the sorted set. In the scenario with the highest computational demand, the algorithm might

need to review each configuration individually to find an appropriate match for each request.

Using the Pareto front derived in the *Offline Phase*, DynaSplit can efficiently execute decisions during runtime, providing solutions that are almost optimal in practical environments. This dual-phase strategy guarantees scalability across various models and hardware configurations, adapting dynamically to diverse workloads.

5.4.2 Applying Configuration

Adjusting the configuration according to the chosen configuration requires modifications at both the edge and the cloud node. At the edge node, the *DynaSplit Controller* initially modifies the CPU and TPU frequencies to align with the settings chosen. In particular, when $TPU_f = off$, the edge hardware accelerator is bypassed; the TPU is fully deactivated to prevent unnecessary power consumption. Additionally, if the head network has not been utilized before, it is loaded.

If $config.L_{network} \neq L$, which implies that the inference task requires cloud processing, *DynaSplit Controller* sends a startup message to the cloud node, specifying which tail network to load and whether to use the GPU for cloud speedup. With respect to both edge and cloud nodes, if needed, five preliminary inference runs are conducted.

5.4.3 Executing Inference

At this point, the inference task can be coordinated. Initially, the head network handles the user's data, leading to the generation of intermediate output. These outputs are then transmitted to the cloud node, where the tail network takes over, processes them further, and returns the finalized results back to the edge node. Ultimately, these inference outcomes are delivered to the user who made the request, thus concluding the request cycle.

To conclude, DynaSplit utilizes a strong approach to address the complex issue of multi-objective optimization. It harnesses NSGA-III for approximating the Pareto front and identifies the optimal configuration based on user-defined latency needs. This process guarantees maximum performance and energy efficiency in edge-cloud infrastructures.

CHAPTER 6

Implementation

DynaSplit has been developed with Python 3.9 running on an edge device, while on the cloud server, Python 3.11 is utilized for its execution.

In order to realize the process of multiobjective optimization, the *DynaSplit Solver*, utilizes Optuna [ASY+19], which is an open-source framework capable of efficiently exploring predefined parameter spaces with capabilities suited to tackle multi-objective optimization problems (MOOPs). The optimization workflow integrates the NSGAIIISampler[1], which executes the NSGA-III algorithm while applying default parameter settings, along with Optuna's results database for effective data management. Furthermore, our empirical analysis involves the use of the GridSampler[2], enabling a near-exhaustive evaluation of the search space. Optuna's robust functionality allows us to target reductions in latency, optimize energy consumption, and achieve high accuracy concurrently, leveraging a configuration space we have previously defined. Through the integration of Optuna's tools, we are able to methodically investigate a variety of possible configurations, ensuring that multiple objectives are simultaneously addressed, thus boosting the overall performance and efficiency of our edge-cloud prototype.

Communication between edge and cloud nodes utilizes gRPC[3] with the type of bidirectional streaming. Initially, we employed synchronous unary-unary calls for this data exchange. However, this approach fell short of capturing precise power usage data during the brief phases of partial inference because the power meters' sampling rates were insufficient. To counteract this issue, we explored making 1,000 asynchronous unary-unary calls to mimic single requests while allowing us to complete all head inferences first. Subsequently, we aim to send 1,000 intermediate results to the cloud before executing all

[1] https://optuna.readthedocs.io/en/latest/reference/samplers/generated/optuna.samplers.NSGAIIISampler.html

[2] https://optuna.readthedocs.io/en/latest/reference/samplers/generated/optuna.samplers.GridSampler.html

[3] https://grpc.io/

tail inferences. This approach would extend the duration of the cumulative head or tail inferences, enabling the power meters to collect sufficient samples during the respective inference periods. Unfortunately, this strategy quickly led to memory depletion on the Raspberry Pi, as it was overwhelmed by the numerous concurrent calls. This limitation arises from gRPC's mechanism, which stores full request data for each open request in memory. To address this shortcoming, we transitioned to bidirectional streaming, which dispatches metadata just once at the start of the stream instead of with every single request and is able to free memory during the streaming of data. Although this method does not perfectly replicate real-world scenarios, it allows us to transmit data efficiently in a continuous stream, minimizing memory usage by progressively clearing intermediate outputs. This feature is essential given the resource constraints on the edge node.

Our prototype is compatible with two neural networks that have been pre-trained using the ImageNet [RDS+15] dataset: VGG16 [SZ15] and the Vision Transformer (ViT) [DBK+21]. VGG16 is executed through the Keras applications available in TensorFlow. In contrast, ViT is operational via a different Keras-based implementation[4].

For executing edge inference via the edge hardware accelerator, in our case the Google Coral USB accelerator[5], the head sections of the VGG16 network are converted to 8 bit integer format and tailored for the TPU through LiteRT[6], previously known as Tensorflow Lite. This conversion, or quantization, was performed using a representative selection of 100 random images drawn from the ImageNet validation dataset. After that, the quantized head models have to be compiled to run on specific hardware[7]. The entire process is depicted in Figure 6.1.

Given memory limitations, ViT cannot execute on the edge TPU and must instead utilize conventional 32-bit floating point operations executed on the CPU. Furthermore, converting the head partitions of the Vision Transformer network into Tensorflow Lite formats was essential. Moreover, to maintain a consistent software configuration irrespective of the given edge processing unit, the non-quantized head networks that are executed on the edge CPU underwent conversion to Tensorflow Lite as well. When working with ViT, it was necessary to approximate the iGELU activation function because TensorFlow Lite does not provide native support for it [RMEZ23].

Inference is conducted on the cloud node within a Docker container configured for GPU usage, using TensorFlow[8], and additional libraries:

- `grpcio==1.66.1`
- `pandas==2.2.2`
- `psutil==6.0.0`

[4] `https://github.com/faustomorales/vit-keras/tree/master/vit_keras`
[5] `https://coral.ai/products/accelerator`
[6] `https://ai.google.dev/edge/litert`
[7] `https://coral.ai/docs/edgetpu/compiler`
[8] tensorflow/tensorflow:2.15.0-gpu

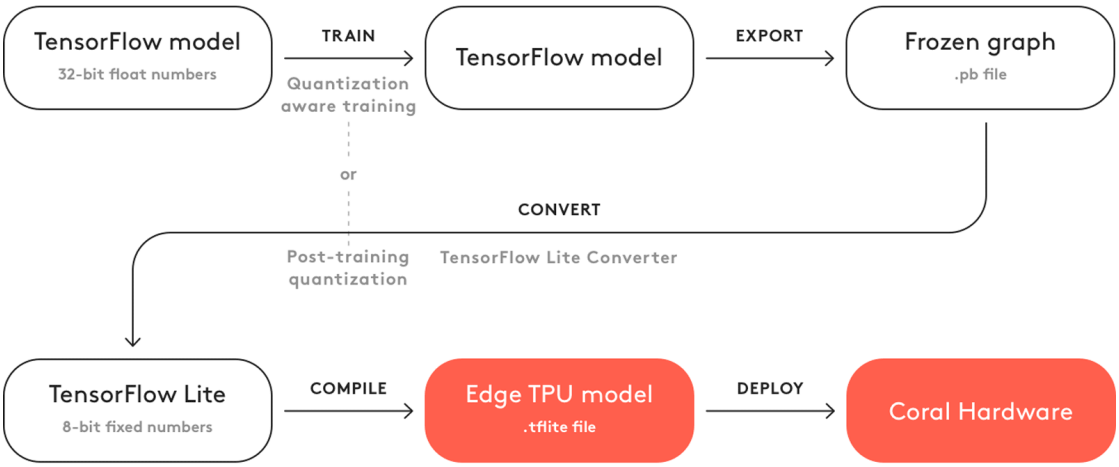


Figure 6.1: The fundamental procedure for developing a model intended for deployment on the edge TPU [Goo24].

- `nvidia-ml-py==12.560.30`
- `scikit-learn==1.5.1`

The software stack operating on the edge device comprises the following components:

- `tensorflow==2.15.0`
- `grpcio==1.66.1`
- `tensorflow-datasets==4.9.3`
- `psutil==6.0.0`

In addition, we used a separate Raspberry Pi to operate the *`DynaSplit` Controller*. This auxiliary node used the following stack of software:

- `optuna==4.0.0`
- `scikit-learn==1.5.2`
- `pymeas==0.2.0`
- `paramiko==3.4.1`
- `pandas==2.2.2`
- `psutil==6.0.0`

CHAPTER 7

Evaluation

We perform an empirical assessment of DynaSplit through a combination of scalability simulations and practical experiments on a real-world edge-cloud setup. In the following sections, we detail the configuration of the experiments (section 7.1), outline our experimental methodology (section 7.2), and present the empirical findings of our practical experiments (section 7.3), where we will also conduct an ablation study to check the effect of the explored fraction of the search space of the *DynaSplit Solver* during the *Offline Phase* (subsection 7.3.4). Then we will also inspect the results of the scalability simulation (section 7.4). We finish with an overhead analysis of our *DynaSplit Controller* (section 7.5).

7.1 Experimental Setup

A realistic edge-cloud testbed is established using an edge node located within the High Performance Computing (HPC) research group's laboratory, combined with a cloud node that is part of the Grid5000 computing infrastructure[1].

In Figure 7.1, the setup utilized for our empirical analysis is displayed; a schematic representation is shown on the left side, while the actual laboratory setup is illustrated on the right.

7.1.1 Edge Node

In the HPC laboratory, the edge node operates using a Raspberry Pi 4 Model B Rev 1.4, featuring Raspberry Pi OS 64-bit, 8 GiB RAM and a quad-core ARMv8 processor. To minimize energy usage, we deactivate non-essential features such as Wi-Fi, Bluetooth, and both the power and activity LEDs. Normally, the CPU frequency adapts dynamically

[1] https://www.grid5000.fr/w/Grid5000:Home

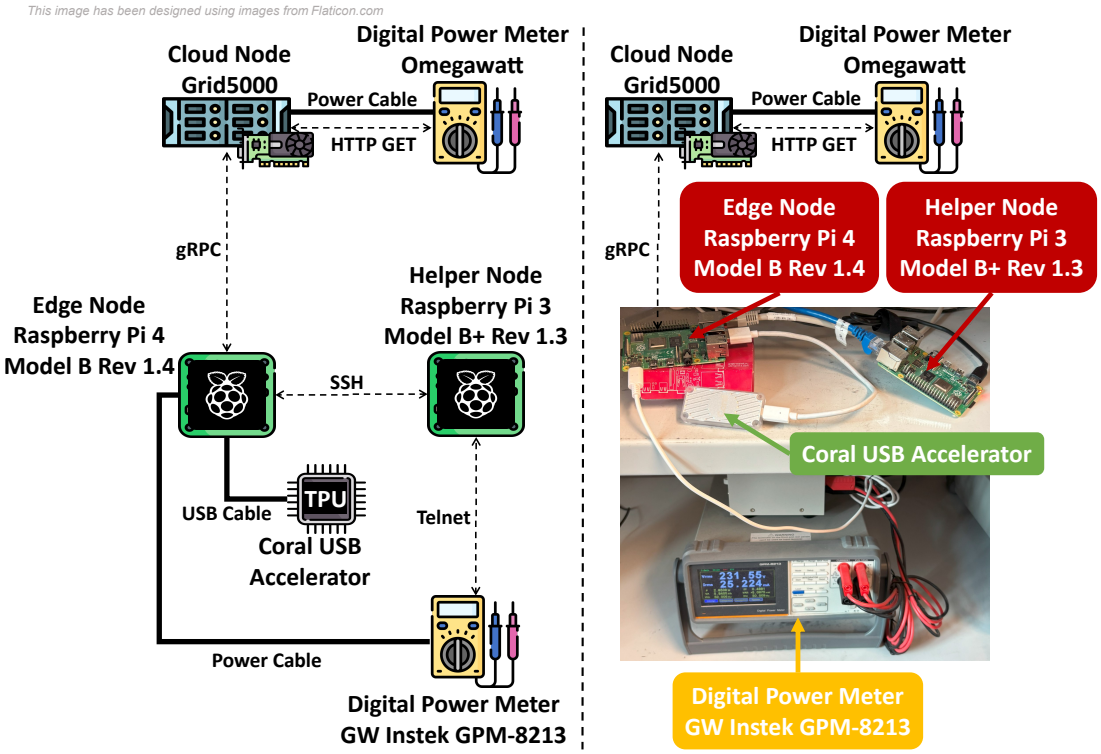


Figure 7.1: Symbolic and actual depiction of the testbed used in out experiments [MTIB24].

from 600 MHz to 1.5 GHz, with cooling measures activated if the CPU temperature surpasses 85°C[2]. However, we configure the CPU to use the `userspace` frequency governor, eliminating dynamic adjustments and granting us control to assign specific frequencies ranging from 0.6 GHz to 1.8 GHz with increments of 0.2 GHz. Using fixed static frequencies is crucial to addressing our optimization challenge, as it provides a stable and uniform frequency parameter necessary for the hardware-software codesign approach. This method allows us to adjust hardware settings in harmony with software improvements, yielding more beneficial compromises between energy efficiency and computational performance. Furthermore, the edge node features a Google Coral USB Accelerator[3] with a 4 TOPS int8 tensor processing unit (TPU) coprocessor. When operated with the `libedgetpu1-std` package, the TPU's clock speed is set to 250 MHz, and it can reach 500 MHz using the `libedgetpu1-max` package. The USB port is disabled when the TPU is not needed in a given configuration, avoiding unnecessary power consumption. We evaluate the active power consumption of the edge node using a

[2] `https://www.raspberrypi.com/documentation/computers/config_txt.html#monitoring-core-temperature`

[3] `https://coral.ai/products/accelerator`

GW Instek GPM-8213 Digital Power Meter[4], which offers a sampling rate of 200 ms and a resolution of 1 mW. We then calculate energy usage through trapezoidal integration[5] of the recorded power metrics.

7.1.2 Cloud Node

The cloud node, part of the Grid5000 cluster, is located in Lyon, France, while the edge node is located in the laboratory of the HPC research group in Vienna. It is equipped with dual Intel Xeon E5-2698 v4 processors, 512 GiB of RAM, and is equipped with 8 NVIDIA Tesla V100 GPUs. However, in our experiments, we use only one GPU. This node was chosen because it has a physical Omegawatt wattmeter[6] connected to it, that records power consumption every 20 ms with an accuracy of 0.1 watt. The capability of directly monitoring power usage at the node level makes this node ideal for our edge-cloud testbed.

7.1.3 Helper Node

In our experiment, a third node is deployed to manage the execution of the `DynaSplit` components, specifically the *`DynaSplit` Solver* and the *`DynaSplit` Controller*. This node is configured using a Raspberry Pi 3 Model B+ Rev 1.3, which operates on Raspberry Pi OS 64-bit and includes 1 GiB of RAM along with a quad-core ARMv8 processor.

7.2 Experimental Plan

We perform two distinct experiments, referred to as the *Testbed Experiment* and the *Simulation Experiment*. Both experiments employ identical evaluation metrics, which are described in subsection 7.2.2, to assess and contrast `DynaSplit` against four baselines methods detailed in subsection 7.2.3.

Specifically, the *Testbed Experiment* evaluates `DynaSplit` in a practical setting by using our edge-cloud testbed to manage a workload comprising 50 user requests per network, as detailed in subsection 7.2.1. During this experiment, we perform an ablation study on the effectiveness of only exploring 20% of the search space in the *Offline Phase*. We do this by examining `DynaSplit`'s efficiency when using non-dominated configurations derived from exploring 20% of the search space via our *`DynaSplit` Solver*, against the non-dominated configurations obtained from a more extensive search covering approximately 80% of the configuration space.

In contrast, the *Simulation Experiment* is designed to evaluate `DynaSplit`'s performance under an increased volume of user requests. Specifically, this involves simulating up to 10,000 user requests using the evaluation metrics derived from previous examinations

[4] `https://www.gwinstek.com/en-global/products/detail/GPM-8213`
[5] `https://scikit-learn.org/dev/modules/generated/sklearn.metrics.auc.html`
[6] `https://www.grid5000.fr/w/Lyon:Wattmetre`

conducted during the search space exploration and the *Testbed Experiment*. We ensured that every configuration incorporated in the simulation went through a minimum of five evaluations on the testbed. Then these configurations were randomly sampled from the pool of recorded observations corresponding to the specified configurations.

Furthermore, we perform an evaluation to examine the impact of execution time associated with the `DynaSplit` *Controller*, as detailed in section 7.5.

7.2.1 Workload Generation

We produce a total of 50 requests for the *Testbed Experiment* and a considerable 10,000 requests for the *Simulation Experiments*. Each request simulates a user that needs to perform an image classification inference task involving 1,000 images taken from the ImageNet [RDS+15] validation dataset.

Table 7.1: Illustration of observed minimum and maximum latency limits for VGG16 and ViT architectures, with their respective configuration [MTIB24].

	Min. Latency		**Max. Latency**	
	Value	*Configuration*	*Value*	*Configuration*
VGG16	90.6 ms	CPU Freq.: 1.2 GHz TPU: No GPU: Yes Split Layer: 0	5,026.8 ms	CPU Freq.: 0.6 GHz TPU: No GPU: No Split Layer: 20
ViT	118.8 ms	CPU Freq.: 1.4 GHz TPU: No GPU: Yes Split Layer: 0	10,287.6 ms	CPU Freq.: 0.6 GHz TPU: No GPU: No Split Layer: 18

To allocate a quality of service (QoS) level to each request, we apply the Weibull distribution, setting its shape parameter to 1, thus transforming it into an exponential distribution. This approach effectively models real-world latency patterns [APMW19]. For each neural network, we draw samples from this distribution, subsequently scaling these samples so that the smallest aligns with the minimum observed latency of the network and the largest corresponds to the maximum observed latency, as seen in Table 7.1. Additionally, Figure 7.2 showcases the distribution of latency values generated for the two networks.

7.2.2 Evaluation Metrics

We perform an assessment of `DynaSplit` with respect to four key metrics, which include latency, the level of QoS violations, the energy consumption, and the accuracy of inference results.

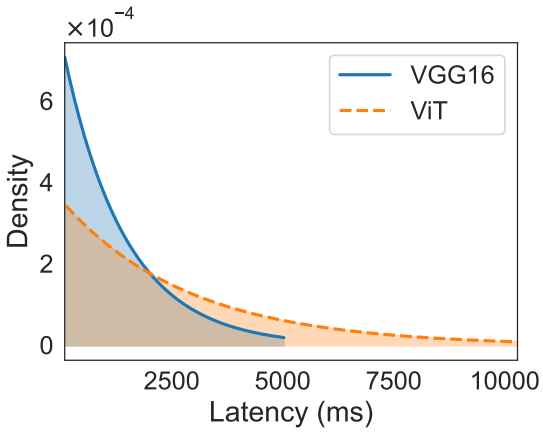


Figure 7.2: Distribution of requests throughout inference duration, specifically for the VGG16 and ViT network architectures [MTIB24].

Latency

We assess latency, expressed in milliseconds, employing Python's `time.perf_counter_ns` function. The overall latency is calculated by summing the following components:

- *Edge Latency* encompasses processes such as image scaling, batch formation, execution of neural network inference, and decoding of the final output.
- *Cloud Latency* involves steps such as the deserialization of intermediary outputs, invoking the tail model, and decoding the final output.
- *Network Latency* represents the residual duration after deducting the edge and cloud latencies, specifically attributing to data transmission time.

QoS Violations

We assess QoS violations by counting the number of requests whose latency exceeds the established quality of service limit and then measure the extent of non-compliance in terms of milliseconds.

Energy Consumption

We separately monitor energy usage in Joules for both edge and cloud processing. For consistent measurement, we batch 1,000 inferences per user inquiry. The edge executes 1,000 initial inferences, after which the results are forwarded to the cloud, which then handles 1,000 concluding inferences. By lengthening the inference period, this approach ensures trustworthy power measurements, surpassing the constraints posed by the power meter's sampling frequency, 200 ms on the edge node and 20 ms on the cloud node.

Accuracy

The measurement is determined by dividing the number of images accurately categorized by the complete count of images. It is important to mention that every user request comprises 1,000 inferences. The metric used to evaluate each request is obtained by averaging the outcomes across these 1,000 inferences.

7.2.3 Baseline Methods

This study focuses on assessing `DynaSplit`'s ability to dynamically select configurations by benchmarking it against four baselines described as follows:

1. *Cloud-Only (cloud)*: In this setup, all inference tasks are executed on the cloud node using the GPU, with the edge CPU operating at its highest frequency.
2. *Edge-Only (edge)*: Here, all inference tasks are handled by the edge node, employing the TPU at its maximum frequency, or switching it off if unused (as in ViT scenarios), while running the edge CPU at its maximum frequency.
3. *Fastest (latency)*: This baseline represents the configuration with the lowest latency, chosen from among the non-dominated configurations identified during the *Offline Phase*.
4. *Energy-Saving (energy)*: This configuration is the most energy efficient, extracted from the non-dominated configurations found in the *Offline Phase*.

7.3 Testbed Experiment Results

In Table 7.2, we can see the Pareto front that was obtained during the *Offline Phase* for the VGG16 network. We observe that the set of optimal configurations includes all types of solutions, cloud, split, and edge. We can also see that the difference in energy usage is high when the cloud node is involved in the computation. The last two configurations are effectively never chosen during the *Online Phase* since we scheduled based on expected latency.

In Table 7.3, the Pareto front for the Vision Transformer network is illustrated. Unlike in the VGG16 configurations, we do not utilize the edge TPU due to memory constraints. Consequently, quantized head models are not needed, leading to consistent accuracy across all configurations. This is why the accuracy metrics are omitted from the table. Furthermore, unlike VGG16, no edge configuration was identified during the *Offline Phase* of the *`DynaSplit` Solver*. Here, we can see that the differences in energy consumption are also more nuanced, which is a result of not having edge-only configurations.

Figure 7.3 illustrates the *`DynaSplit` Controller*'s scheduling choices for each neural network. The selected configurations are classified according to the split layer into cloud-only, edge-only, or split execution. In the case of VGG16, the schedule includes 37

#	**Objectives**			**Configuration**				**Type**
	Lat. (ms)	*Eng. (J)*	*Acc.*	*CPU (MHz)*	*TPU*	*GPU*	*Layer*	
1	442.42	2.17	0.603	800	std	No	22	edge
2	428.54	2.25	0.603	1200	max	No	22	edge
3	489.37	60.89	0.607	1400	std	No	21	split
4	120.88	61.58	0.609	1600	max	Yes	17	split
5	115.18	61.83	0.611	1400	std	Yes	15	split
6	112.96	64.15	0.608	1400	std	Yes	14	split
7	110.98	64.86	0.608	1600	std	Yes	14	split
8	90.59	64.99	0.605	1200	off	Yes	0	cloud
9	109.69	65.16	0.609	1200	std	Yes	10	split
10	112.01	65.80	0.611	1400	max	Yes	15	split

Table 7.2: The Pareto front obtained with the `DynaSplit` *Solver* for VGG16, already sorted according to `DynaSplit`'s rules.

#	**Objectives**		**Configuration**				**Type**
	Lat. (ms)	*Eng. (J)*	*CPU (MHz)*	*TPU*	*GPU*	*Layer*	
1	4371.71	81.35	1600	off	Yes	16	split
2	3994.43	83.79	1800	off	No	17	split
3	3300.06	85.60	1200	off	Yes	11	split
4	2059.02	85.85	1800	off	Yes	10	split
5	1433.70	87.68	800	off	Yes	8	split
6	1111.61	88.10	1800	off	Yes	7	split
7	933.59	88.31	1400	off	Yes	6	split
8	213.86	89.22	1200	off	Yes	4	split
9	187.49	91.32	600	off	Yes	3	split
10	145.25	91.43	1400	off	Yes	4	split
11	139.06	92.10	1400	off	Yes	3	split
12	118.80	93.55	1400	off	Yes	0	cloud

Table 7.3: The Pareto front achieved through the `DynaSplit` *Solver* for ViT, sorted following `DynaSplit`'s criteria.

requests for edge computing, just two for cloud computing, and 11 for split execution. In contrast, for ViT, just a single request is allocated for cloud execution, while the remaining 49 requests are scheduled for split execution. Importantly, the absence of edge computing for ViT stems from the `DynaSplit` *Solver*'s inability to pinpoint any edge-exclusive setups during the *Offline Phase*. The frequent reliance of VGG16 on edge computing is attributed to its compatibility with the edge accelerator, in contrast to ViT.

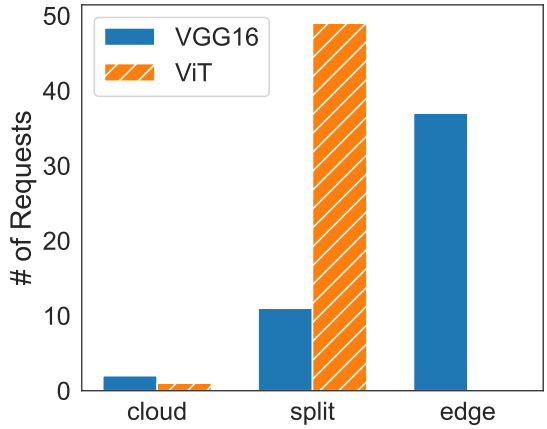
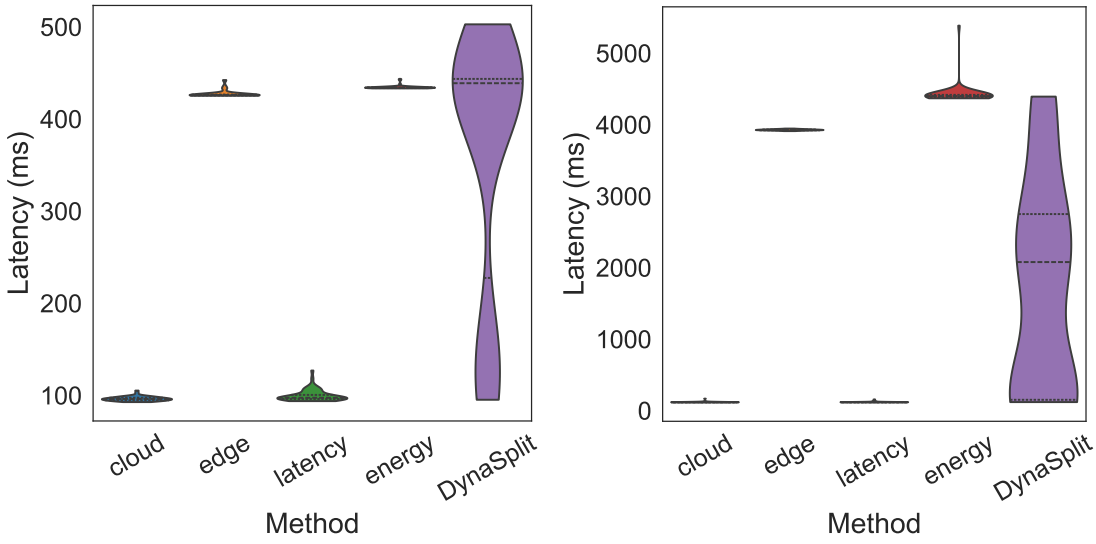


Figure 7.3: This image depicts the scheduling choices executed by `DynaSplit` [MTIB24].



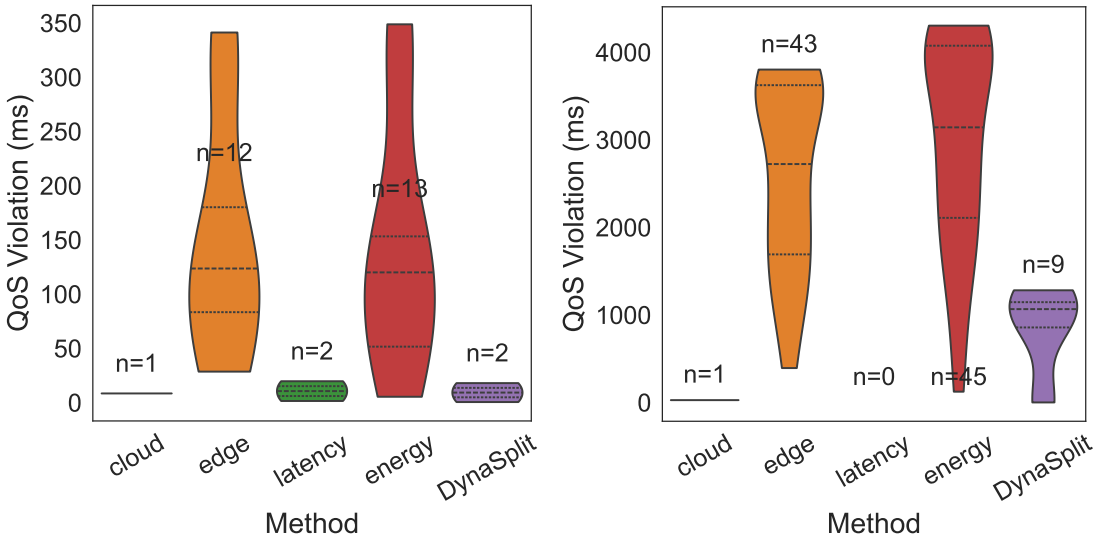
(a) VGG16 (b) ViT

Figure 7.4: Illustration of latency distributions for the VGG16 and ViT architectures [MTIB24].

7.3.1 Latency & QoS Violations

In Figure 7.4, the latency distributions for 50 requests are evaluated for four static baselines: cloud, edge, latency, and energy, along with `DynaSplit`. Within our study's violin plots, quartiles are denoted by horizontal lines, and the violin's form illustrates the distribution density. Regarding the VGG16 model, the cloud and latency baselines capture medians of 96 ms and 97 ms, respectively, while the edge and energy baselines yield medians of 425 ms and 434 ms, respectively. For the ViT model, the cloud and

latency medians are 117 ms, whereas edge and energy baselines show more pronounced latencies of 3,926 ms and 4,400 ms, respectively. The DynaSplit method dynamically adjusts latency, striking a balance predominantly near the edge baseline for VGG16 and close to 2,000 ms for ViT. Significantly, for the ViT model, DynaSplit offers a nuanced range of latencies that smoothly transitions between edge and cloud baselines, a contrast to the more stepwise adjustments as with VGG16.



(a) VGG16 (b) ViT

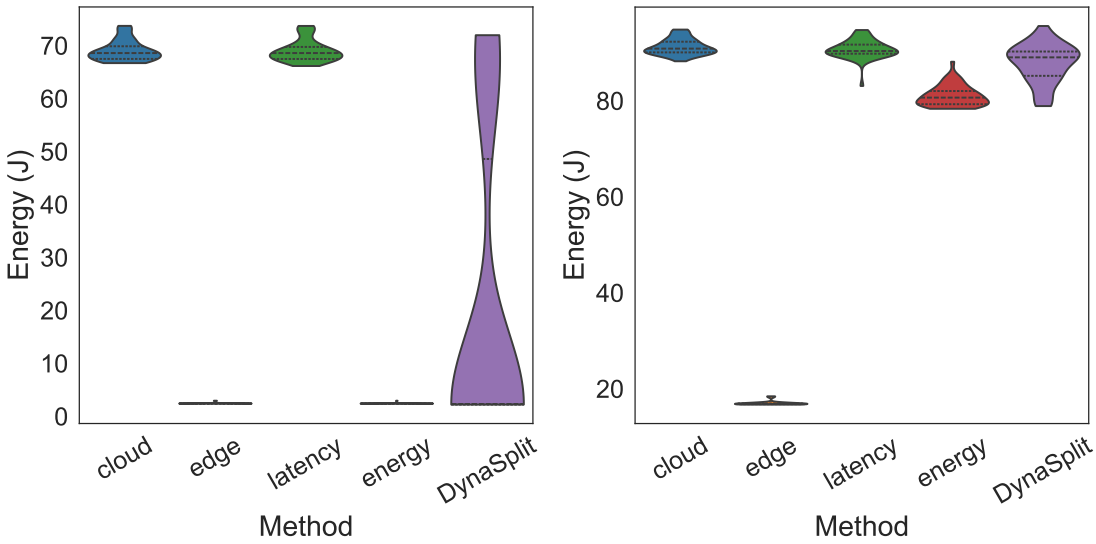
Figure 7.5: Illustration depicting the QoS violation patterns exhibited by VGG16 and ViT models [MTIB24]. The variable n represents the total count of violations.

In Figure 7.5, the QoS violation distributions are depicted. Each violin plot reveals the degree to which the requests that failed to meet their quality of service deadlines exceeded the specified time threshold. Both networks exhibit similar patterns. For the cloud and latency baselines, the QoS levels were violated at most twice with very minor exceedance: less than 20 milliseconds for VGG16 and less than 27 milliseconds for ViT. In contrast, the edge and energy baselines showed more significant violations. Approximately 25% of the requests in VGG16, and as much as 90% in ViT, exceeded their deadlines, the median exceedance reaching 120 milliseconds for VGG16 and about 3,000 milliseconds for ViT. The DynaSplit approach produced violations in 4% of requests for VGG16 and 18% for Vision Transformer, with median exceedances of around 10 milliseconds for VGG16 and 1,000 milliseconds for ViT.

7.3.2 Energy Consumption

In Figure 7.6, energy consumption is depicted. The cloud and latency baselines consistently show higher energy usage compared to other baselines and DynaSplit, with median values reaching 68 J for VGG16 and exceeding 90 J for Vision Transformer. In

contrast, for VGG16, both the edge and energy baselines maintain a median energy usage below 3 J. Regarding ViT, the median energy usage for the edge baseline is 16 J, while the energy baseline, being a split configuration (i.e., the most energy-efficient among the non-dominated configurations), records a median of 80 J. This is again due to the fact that the *DynaSplit Solver* did not explore any edge configurations during the *Offline Phase*. DynaSplit showcases adaptive capabilities, achieving a median energy consumption for VGG16 that aligns with the edge and energy baselines at under 3 J, although it can peak at 72 J. In the case of ViT, DynaSplit aligns more closely with the cloud, energy, and latency baselines, displaying a median energy consumption of 89 J.



(a) VGG16 (b) ViT

Figure 7.6: Illustration of how energy usage is distributed for the VGG16 and Vision Transformer (ViT) networks [MTIB24].

7.3.3 Accuracy

We detected very slight differences in accuracy, less than 1%. Configurations utilizing the cloud, which operate the entire model in 32-bit floating point without employing quantization, show marginal benefits. Nevertheless, our method achieves strong performance, revealing that it maintains comparable accuracy to baseline approaches without any detrimental effects.

7.3.4 DynaSplit Search vs. $\sim 80\%$ Search

Our DynaSplit method, which investigates 20% of the search space equal to 184 trials, is evaluated against an exploration of the 81.5% search space, approximated to 80% for simplicity, which involved 747 trials for the VGG16 network. In Table 7.4, we can see the Pareto front obtained through the exhaustive exploration of the search space. We

observe that there are now more non-dominated configurations (15) compared to our DynaSplit approach (10). Furthermore, we notice that there are significantly more edge-only configurations with varying edge parameters.

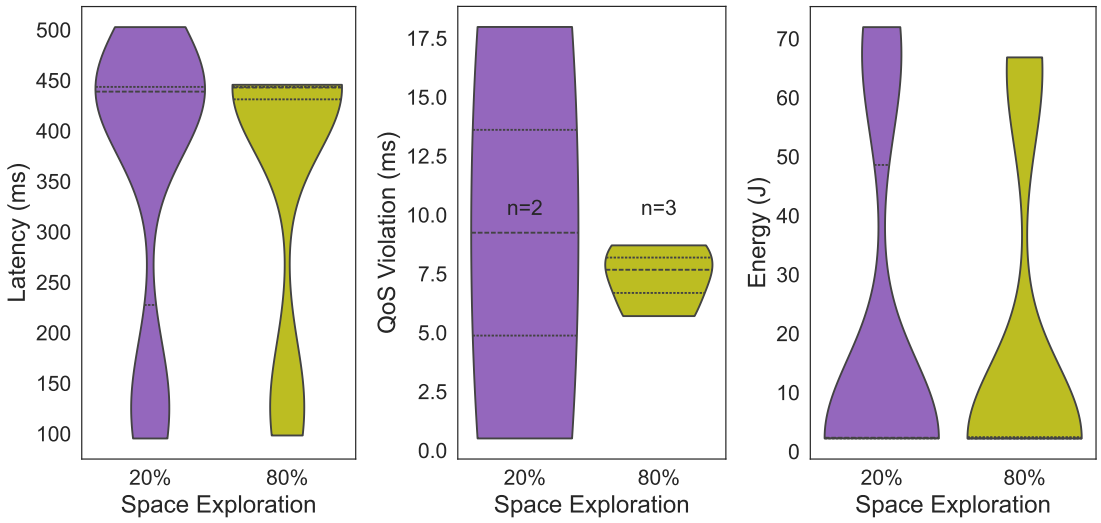
#	**Objectives**			**Configuration**				**Type**
	Lat. (ms)	*Eng. (J)*	*Acc.*	*CPU (MHz)*	*TPU*	*GPU*	*Layer*	
1	442.42	2.17	0.603	800	std	No	22	edge
2	438.65	2.23	0.603	1000	std	No	22	edge
3	431.01	2.24	0.603	1000	max	No	22	edge
4	428.54	2.25	0.603	1200	max	No	22	edge
5	426.46	2.27	0.603	1400	max	No	22	edge
6	424.52	2.32	0.603	1800	max	No	22	edge
7	1751.16	7.85	0.604	1800	off	No	22	edge
8	127.23	59.61	0.609	800	max	Yes	17	split
9	115.66	60.82	0.609	1800	max	Yes	18	split
10	115.18	61.83	0.611	1400	std	Yes	15	split
11	100.54	62.43	0.609	1600	max	Yes	10	split
12	114.55	64.32	0.611	1400	max	Yes	15	split
13	108.58	64.51	0.611	1800	max	Yes	15	split
14	90.59	64.99	0.605	1200	off	Yes	0	cloud
15	100.47	65.13	0.609	1800	max	Yes	10	split

Table 7.4: The Pareto front discovered through an exhaustive search for VGG16, sorted according to DynaSplit's rules.

The empirical findings reveal that the *DynaSplit Controller* scheduled the same amount of requests for computations executed solely in the cloud. For scenarios involving split computation and computations conducted entirely at the edge, the variations were minimal, with only one data point differing. The exhaustive version scheduled one request more for edge-only computation and one less for split computation compared to our DynaSplit approach.

We conducted a comprehensive analysis to understand the exploration's influence on evaluation metrics. As illustrated in Figure 7.7, both methodologies yielded comparable results, showing no notable discrepancies in terms of latency, quality of service (QoS) violations or energy usage. Observing the latency more closely (Figure 7.7a), it is evident that each method manages most requests within the specified time limits. Although latency readings of around 500 ms appear occasionally in the 20% exploration, which are absent in the 80% exploration, these are exceptions and have minimal influence on meeting deadlines. Furthermore, Figure 7.7b indicates that both approaches result in a very low number of QoS violations (fewer than 3), where violations miss the target by a maximum of just 20 ms. Insignificant variations are apparent in energy consumption, as depicted in Figure 7.7c.

In summary, a modest 20% exploration of the search space suffices, as it produces



(a) Latency (b) QoS Violation (c) Energy

Figure 7.7: Comparison of latency, QoS violations, and energy usage for VGG16 between exploration levels of 20% and 80% [MTIB24].

performance almost equivalent to exploration of 80% of the configuration space, without any apparent drawbacks.

7.4 Simulation Experiment Results

In Figure 7.8, the allocation of incoming requests by `DynaSplit` across cloud, split, and edge computations for the VGG16 and ViT networks is depicted, considering up to 10,000 simulated requests. The proportion of requests directed towards cloud computation is minimal, with VGG16 at 4% and ViT at 1%, reflecting the results of the *Testbed Experiment*. For ViT, there is an absence of requests scheduled solely for edge computation since the non-dominated configuration set (see Table 7.3) lacks such configurations. In contrast, VGG16 opts for more split configurations, likely influenced by its more granular variations of the QoS level, allowing *`DynaSplit` Controller* to exploit a broader array of configurations. Consequently, scheduling decisions are evenly divided between split and edge configurations, with 4857 and 4695 requests, respectively.

7.4.1 Latency & QoS Violations

In Figure 7.9, the latency distribution in VGG16 (see Figure 7.9a) and Vision Transformer (see Figure 7.9b) is illustrated. As observed in the *Testbed Experiment*, the cloud and latency baselines consistently deliver reduced latencies, with medians of $96 - 97$ ms for VGG16 and $117 - 118$ ms for ViT, relative to the higher figures found in the edge and energy baselines, which have medians ranging from $425 - 442$ ms for VGG16 to $3926 - 4400$

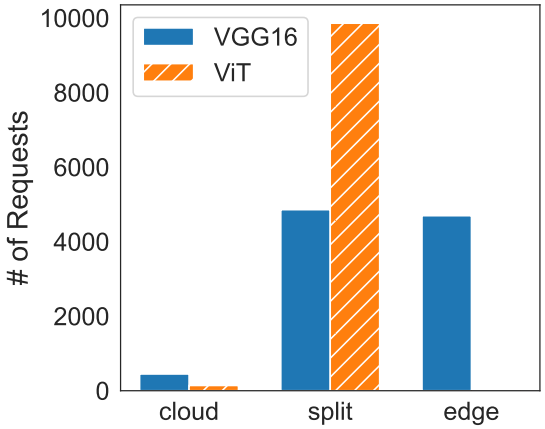
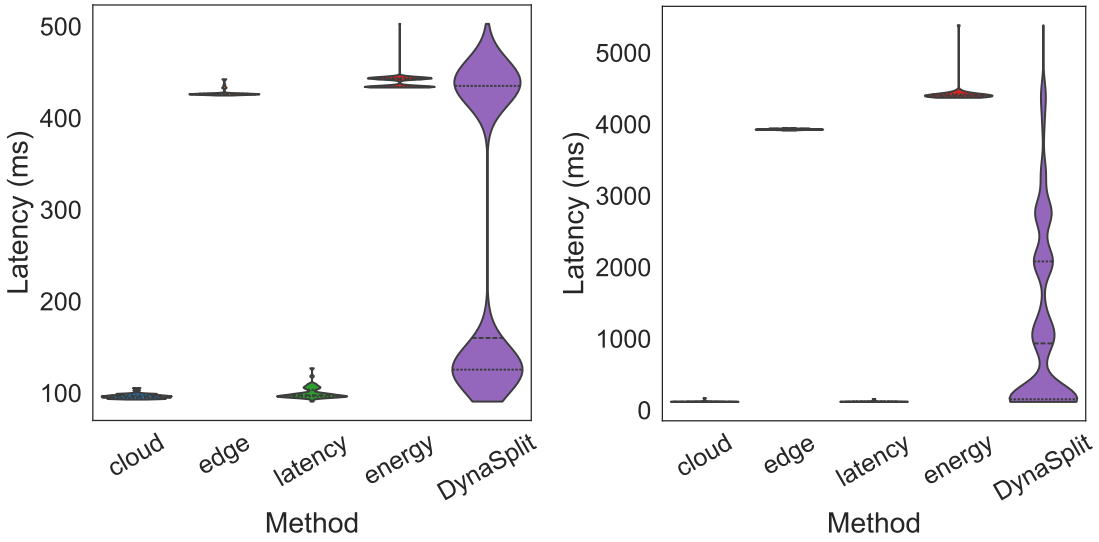


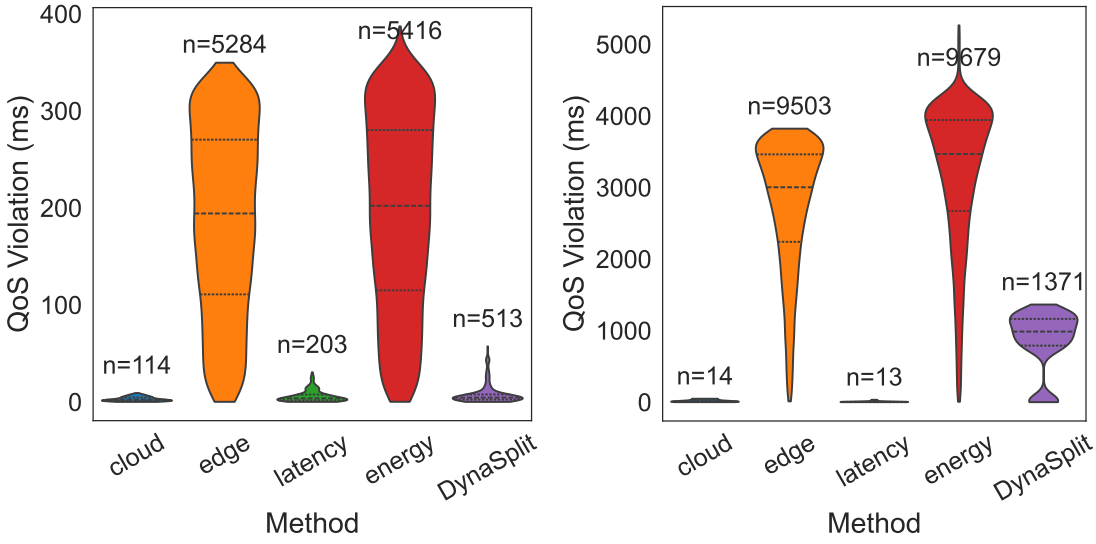
Figure 7.8: Illustration of `DynaSplit` scheduling choices made throughout the *Simulation Experiment* concerning both the VGG16 and ViT network architectures [MTIB24].



(a) VGG16 (b) ViT

Figure 7.9: Depicted here is the latency distribution for both the VGG16 and ViT network models, as measured within the context of the *Simulation Experiment* [MTIB24].

ms for ViT. The `DynaSplit` framework results in a divided latency distribution for VGG16, spanning cloud and edge latencies, which achieves a median latency of 160 ms, which is lower than in the *Testbed Experiment*, attributed to a greater allocation of split computation requests. For ViT, `DynaSplit` presents a spread of latency from the edge to the cloud baselines, showing a concentrated distribution at cloud latencies with a median latency of 933 ms.



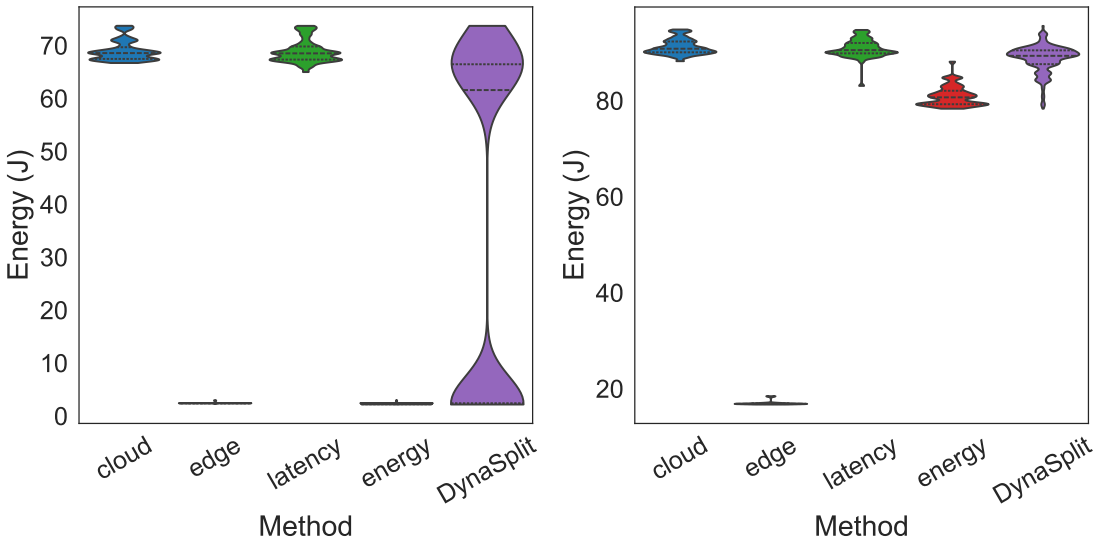
(a) VGG16 (b) ViT

Figure 7.10: Depicted here are the QoS violation patterns for the VGG16 and ViT architectures as observed throughout the *Simulation Experiment* [MTIB24].

As seen in Figure 7.10, both the cloud and latency baselines result in minimal QoS violations, staying within a 2% threshold for VGG16 and 0.1% for Vision Transformer, while median exceedances reach a peak of 3 ms for VGG16 and 10 ms for ViT. In contrast, the edge and energy baselines exhibit substantially higher violation rates, up to 54% for VGG16 and as much as 96% for ViT, with median exceedances increasing to 202 ms for VGG16 and 3467 ms for ViT. On the other hand, `DynaSplit` exhibits about 5% violations for VGG16 and 14% for Vision Transformer, with median exceedances at 4 ms for VGG16 and 986 milliseconds for ViT. These findings are in line with the results of the *Testbed Experiment*.

7.4.2 Energy Consumption

Referring to Figure 7.11, it is shown that energy consumption is consistently elevated for the cloud and latency baselines, with median values of 69 J for VGG16 and 91 J for Vision Transformer. In contrast, the edge and energy baselines demonstrate significantly lower energy usage for VGG16, marked by a median of 2 J. For ViT, the absence of an edge configuration found by the *`DynaSplit` Solver* during the offline phase results in the energy baseline showing a higher median of 81 J, as opposed to the edge baseline's median of 17 J. When examining the `DynaSplit` methodology for VGG16, there is a concentration of data at cloud and edge-comparable energy levels, with a median energy use of 62 J, greater compared to the *Testbed Experiment* due to increased scheduling of split decisions. In the case of ViT, `DynaSplit` demonstrates cloud-level energy consumption, reflected by a median of 89 J, consistently indicating lower energy consumption, in



(a) VGG16 (b) ViT

Figure 7.11: Distribution of energy usage for both VGG16 and ViT networks is shown during the *Simulation Experiment* [MTIB24].

accordance with results from the *Testbed Experiment*.

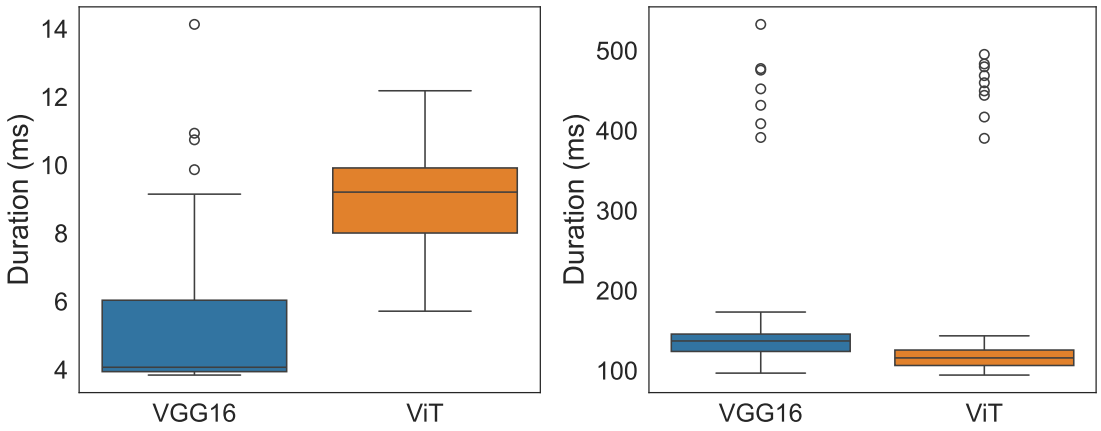
7.4.3 Accuracy

In both the *Testbed Experiment* and the *Simulation Experiment*, we found that the differences in accuracy were minimal, not exceeding 1%. Moreover, `DynaSplit` demonstrates performance that is at least on par with edge computation. It sustains an accuracy level similar to the baseline models without noticeable degradation.

7.5 Overhead Analysis

In our assessment of the run-time overhead associated with the *`DynaSplit` Controller*, we focused on evaluating latency caused by its operational activities and the impact on memory consumption. These measurements were gathered during the execution of the *Testbed Experiment*.

Upon startup, the *`DynaSplit` Controller* executes a one-time loading and sorting of the non-dominated configuration set. Our measurements show that the median time for this process is about 4.2 seconds, with memory consumption reaching a peak of 20 MB. In contrast, when loading the non-dominated configuration set derived from exploring approximately 80% of the VGG16 search space, the operation consumed 29 seconds and used 70 MB of memory.



(a) Selecting Configuration (b) Applying Configuration

Figure 7.12: Illustration of the `DynaSplit` *Controller*'s overhead concerning both the selection and implementation of configurations for the VGG16 and ViT models [MTIB24].

Once the initial setup is complete, the `DynaSplit` *Controller*'s primary responsibility is to identify the optimal configuration for each incoming request. Figure 7.12a illustrates that this process can take up to 12 ms. For VGG16, the median duration is under 5 ms, whereas for Vision Transformer, it is 10 ms. The duration is largely dependent on the number of non-dominated configurations available, with our examples having 15 for VGG16 and 12 for ViT (see Table 7.2 and Table 7.3), respectively.

The operation that requires the most time is the implementation of the chosen configuration, illustrated in Figure 7.12b. Typically, this process is completed in under 200 ms. However, there are occasional deviations, where it can reach 500 ms. Despite these anomalies, the median duration observed for both networks remains below 150 ms.

We evaluate the impact on inference time by comparing the typical overhead duration with the typical edge latencies. For the VGG16 model, which has a median edge latency of 426 ms, choosing the configuration adds an additional 0.96% to the latency, while implementing the configuration results in a 32.14% increase in latency. In the case of the Vision Transformer model, with a 3,922 ms median edge latency, selecting the configuration increases the latency by 0.23%, while applying the configuration causes a 2.95% increase in latency.

The findings indicate that choosing the configuration incurs only a slight additional cost, whereas implementing the configuration accounts for a significant portion of the total overhead but remains minor in relation to the entire inference duration.

7.6 Limitations

Here, we examine the possible limitations of our methodology, emphasizing the specific conditions that might influence our results in practical applications. These factors provide context for our findings and indicate directions for future enhancements.

7.6.1 Offline Phase and Configuration Space Changes

Our strategy incorporates an offline optimization stage designed to estimate and achieve nearly optimal settings for hardware and software configurations. However, if there are substantial changes to the model or hardware setup, this stage needs to be revisited, since the current optimizations may no longer suit the altered conditions. For instance, the inclusion of a new model architecture or the utilization of different hardware elements necessitates re-executing the optimization process to identify updated solutions. This reliance on an offline phase can restrict flexibility and might reduce the system's ability to quickly adapt to new setups. In ever-changing environments where hardware or software undergoes frequent updates, the requirement for recurring offline optimization could potentially introduce added expenses and delays.

7.6.2 Deployment Strategy

Within our experimental framework, we assume a continuously operational cloud server, with the model pre-loaded and warm-up inference computations pre-computed, to ensure latency is minimized during inference. However, this set-up might not fully capture all practical deployment scenarios. In real-world situations, serverless or containerized cloud services may be employed, which allocate resources only when needed. These services typically experience cold-start latency and introduce additional warm-up overhead, consequently affecting both latency and cost. In addition, the ongoing expenses associated with a continuously operational cloud server may be prohibitive for some applications. Although our setup is designed to minimize these factors to maintain experimental uniformity, future research could investigate deployment strategies that are better aligned with environments where resources vary or costs are a concern.

7.6.3 Model Size and Edge Optimization

The selection of models for our experiments reflects their varying fit for execution on either edge or cloud platforms, guided by their resource needs. Models with lighter demands, such as ResNet50 and MobileNetV2, typically perform efficiently on edge devices, offering quicker inference and lower power use. Hence, in these scenarios, split computing does not yield notable gains, as the entire computation can be efficiently managed by the edge device alone. In contrast, larger models such as Vision Transformer are better suited for cloud-based execution, where the powerful GPU in the cloud can speed up processing, although with higher energy consumption. Thus, split computing tends to provide greater benefits for larger models that are more aligned with resource-intensive settings, while

for smaller, resource-efficient networks, configurations limited to edge execution are often more advantageous.

7.6.4 Overhead of Configuration Changes and Scheduling

Altering hardware configurations, such as tweaking CPU frequencies or disabling TPUs, implicates processing overhead. The exact time involved depends on implementation details, possibly requiring complex changes like OS library adjustments if DVFS is not directly applicable. Moreover, scheduling involves computational costs due to the necessity of identifying suitable settings for every request from an available set of non-dominated configurations. Although this overhead is minor, it can influence system efficacy and reaction times, especially in contexts with tight latency constraints or high request traffic. A promising strategy might involve grouping user inquiries according to request type, QoS requirements, and user characteristics, thereby minimizing frequent configuration shifts and easing decision-making loads. Alternatively, employing predetermined settings for distinct request categories might reduce runtime adaptations, thereby boosting efficiency where rapid reconfiguration presents significant challenges.

CHAPTER 8

Conclusions and Future Directions

This chapter summarizes the key findings and contributions of this thesis, addresses research questions, and demonstrates the effectiveness of the DynaSplit framework. It concludes with future research directions aimed at enhancing the framework's capabilities and exploring new applications.

8.1 Conclusion

This study focuses on the issue of implementing machine learning models on edge devices that have limited resources, specifically aiming to optimize both energy efficiency and performance. We introduce DynaSplit, a framework that uses split computing and the co-design of hardware and software, to effectively manage the complexities of determining appropriate split layers and hardware settings by efficiently exploring vast configuration spaces. Our methodology employs a dual-phase strategy that incorporates offline optimization followed by online scheduling, validated through experimentation on an actual testbed deploying pre-trained neural networks. The findings reveal that DynaSplit is capable of lowering energy usage while maintaining strict latency constraints. In detail, empirical analysis demonstrates the effectiveness of our proposed method for scheduling inference requests within edge-cloud systems. Through the use of split computing strategies and the optimization of relevant hardware settings, we can efficiently handle both latency and energy usage without compromising accuracy. This ensures that quality of service (QoS) standards are met with negligible runtime overhead. The results reveal that the DynaSplit approach can achieve a reduction in energy consumption of up to 72% compared to a purely cloud-based processing model, while consistently satisfying approximately 90% of the predefined latency limitations for requests.

Answering the Research Questions

The research questions addressed in this thesis are as follows:

RQ_1: What are the appropriate strategies for splitting DNNs used in image classification?

RQ_2: How do hardware parameters influence the trade-offs between energy consumption, accuracy, and latency in DNN inference?

RQ_3: How can we jointly optimize model splitting and hardware configurations to meet energy and latency requirements for DNN inference?

RQ_1: Appropriate strategies for splitting DNNs. This question is explored in the literature review (see section 2.4.2, section 2.5, chapter 3). The layer-wise splitting of DNNs has been identified as an effective strategy for image classification tasks, as evidenced by previous work such as JointDNN [EAP21], which demonstrated its positive impact on both energy consumption and latency.

RQ_2: Influence of hardware parameters on trade-offs. The preliminary study in section 5.1 addresses this question. Our findings show that increasing CPU frequency reduces both latency and energy consumption on edge devices, although with diminishing returns. The use of TPU and GPU significantly lowers latency and energy consumption, while edge TPU usage minimally affects accuracy. Additionally, the selection of the split layer has a non-linear impact on both latency and energy consumption for edge and cloud devices.

RQ_3: Joint optimization of model splitting and hardware configurations. This question represents the main contribution of this thesis and is addressed by the `DynaSplit` framework, described in chapter 5 and evaluated in chapter 7. `DynaSplit` employs a dual-phase strategy: an offline phase to explore the configuration space using multi-objective optimization and an online phase to dynamically schedule inference requests. This approach enables `DynaSplit` to balance energy consumption and latency while adhering to the QoS requirements with negligible runtime overhead.

In summary, this thesis demonstrates that the integration of split computing with hardware-software co-design offers a powerful solution to deploy DNN models on resource-constrained edge devices, achieving significant improvements in energy efficiency and performance.

8.2 Future Directions

Although `DynaSplit` demonstrates significant potential in optimizing split computing for edge inference, several avenues for future research emerge from its limitations and unexplored opportunities. These directions aim to address the constraints of the current implementation and explore new use cases and enhancements to the framework.

8.2.1 Dynamic Optimization and Adaptability

Future work could focus on improving DynaSplit's adaptability to changes in the underlying configuration space. Dynamic optimization techniques could be explored to enable real-time reconfiguration without the need for extensive offline computations. For instance, integrating reinforcement learning or online optimization algorithms could allow the framework to adapt to changes in hardware, software, or model architectures dynamically, reducing the dependency on pre-computed configurations.

8.2.2 Scalability and Multi-User Environments

DynaSplit has primarily been evaluated in single-user scenarios. Expanding its applicability to multi-user environments, where multiple users or applications share edge and cloud resources, presents an exciting challenge. Investigating resource allocation strategies, such as fair scheduling or priority-based scheduling, could ensure consistent QoS delivery across users while minimizing conflicts in shared environments. This direction could also include studying the impact of increased request traffic on system performance and exploring load-balancing techniques for scalability.

8.2.3 Integration with Serverless Architectures

As discussed in the limitations, serverless architectures introduce unique challenges due to cold start delays and resource allocation overheads. Future research could evaluate DynaSplit's efficiency in serverless deployments, incorporating mechanisms to predict and mitigate cold starts or leveraging warm container pools to maintain low-latency responses. Additionally, optimizing the framework to align with pay-as-you-go cloud pricing models could further enhance its cost-effectiveness.

8.2.4 Energy Efficiency in Heterogeneous Edge Networks

Although DynaSplit demonstrates energy savings for individual inference requests, further research could explore its application in heterogeneous edge networks with varying hardware capabilities. For example, integrating the framework with energy-aware routing and workload distribution mechanisms could improve overall system efficiency in distributed deployments. This line of work could also examine the trade-offs between energy savings and performance in resource-constrained IoT environments.

8.2.5 Broader Model Applicability and Fine-Grained Splitting

Finally, the current implementation focuses mainly on specific model architectures, such as VGG16, ResNet50, MobileNetV2, and ViT. Future studies could extend DynaSplit to support a wider range of models, including those with novel architectures, such as generative models. Furthermore, fine-grained splitting techniques, where model layers are partitioned into smaller computational units, could further enhance flexibility and resource utilization.

Overview of Generative AI Tools Used

I declare that AI tools were used only as supplementary aids in this thesis, with my authorship prevailing. GPT-4o and TeXGPT were used exclusively to check spelling, grammar and refining sentence structure, with my own written text as input in each case. No content was purely generated from a prompt and copied without significant modifications.

List of Figures

List of Tables

List of Algorithms

Acronyms

AI artificial intelligence. xiii, 1, 2, 6, 7, 10–15, 17, 19

AR augmented reality. 9, 10

ARM Advanced RISC Machine. 33, 49, 51

CNN convolutional neural network. 12, 21, 34, 39

CPU central processing unit. xiii, 3, 4, 14, 15, 22, 23, 28, 33–35, 37–39, 43, 46, 49, 50, 52, 54, 66, 68, 73

DAG directed acyclic graph. 22

DL deep learning. 11

DNN deep neural network. 1–5, 11, 12, 14–18, 22–24, 68

DVFS dynamic voltage and frequency scaling. 2, 4, 14, 24, 66

FLOP floating-point operation. 15

GHz gigahertz. 50, 52

GiB gibibyte. 33, 49, 51

GOPS giga operations per second. 14

GPU graphics processing unit. 2–4, 14, 15, 28, 33–36, 38, 39, 43, 51, 52, 54, 65, 68, 73

HPC High Performance Computing. 49, 51

IaaS Infrastructure as a Service. 7

ILP integer linear programming. 22

IoT Internet of Things. 2, 8–10, 17

J Joule. 57, 58, 62

LED light-emitting diode. 49

LTE long-term evolution. 41

MB megabyte. 63

MHz megahertz. 34, 35, 50

ML machine learning. 6, 10, 11, 13, 67

MOOP multi-objective optimization problem. 30, 38, 40, 45

ms milliseconds. 51–53, 56, 57, 59–62, 64

mW milliwatt. 51

NN neural network. xiii, 3, 11, 17, 19, 27, 28, 33, 36, 38, 39, 46, 52–54

NSGA non-dominated Sorting Genetic Algorithm. 40, 43, 45

OS operating system. 49, 51, 66

PaaS Platform as a Service. 7

QoS quality of service. xiii, 2, 3, 5, 14, 23, 28, 36, 38, 41, 42, 52, 53, 57, 59, 60, 62, 66–69, 74

RAM random-access memory. 33, 49, 51

ReLu Rectified Linear Unit. 11

RNN recurrent neural network. 12, 23

RPC remote procedure call. 45, 46

SaaS Software as a Service. 7

SLA service-level agreement. 14, 41

SNN spiking neural network. 22

TOPS tera-operations. 50

TPU tensor processing unit. 2, 4, 14, 22, 33–36, 38, 39, 43, 46, 47, 50, 52, 54, 66, 68, 73

UAV unmanned aerial vehicle. 24

USB Universal Serial Bus. 50

ViT Vision Transformer. 33, 34, 39, 46, 52–58, 60–65, 69, 73–75

Bibliography

[AFG+10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andy Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, 2010.

[AGH20] Ahsan Adeel, Mandar Gogate, and Amir Hussain. Contextual deep learning-based audio-visual switching for speech enhancement in real-world environments. *Inf. Fusion*, 59:163–170, 2020.

[APMW19] Muhammad Asad Arfeen, Krzysztof Pawlikowski, Don McNickle, and Andreas Willig. The role of the weibull distribution in modelling traffic in internet access and backbone core networks. *J. Netw. Comput. Appl.*, 141:1–22, 2019.

[ARAD21] I Nyoman Gede Arya Astawa, Made Leo Radhitya, I Wayan Raka Ardana, and Felix Andika Dwiyanto. Face images classification using VGG-CNN. *Knowl. Eng. Data Sci.*, 4(1):49–54, 2021.

[ASY+19] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[AZTS18] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. Mobile edge computing: A survey. *IEEE Internet Things J.*, 5(1):450–465, 2018.

[BCCN18] Simone Bianco, Rémi Cadène, Luigi Celona, and Paolo Napoletano. Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 6:64270–64277, 2018.

[BVP+21] Amin Banitalebi-Dehkordi, Naveen Vedula, Jian Pei, Fei Xia, Lanjun Wang, and Yong Zhang. Auto-split: A general framework of collaborative edge-cloud AI. In Feida Zhu, Beng Chin Ooi, and Chunyan Miao, editors, *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, pages 2543–2553. ACM, 2021.

[BVS13] Rajkumar Buyya, Christian Vecchiola, and Thamarai Selvi Somasundaram. *Mastering Cloud Computing: Foundations and Application Programming.* Morgan Kaufmann, 05 2013.

[BWL20] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *CoRR*, abs/2004.10934, 2020.

[CB18] Hyomin Choi and Ivan V. Bajic. Deep feature compression for collaborative object detection. In *2018 IEEE International Conference on Image Processing, ICIP 2018, Athens, Greece, October 7-10, 2018*, pages 3743–3747. IEEE, 2018.

[CCB20] Robert A. Cohen, Hyomin Choi, and Ivan V. Bajic. Lightweight compression of neural network feature tensors for collaborative intelligence. In *IEEE International Conference on Multimedia and Expo, ICME 2020, London, UK, July 6-10, 2020*, pages 1–6. IEEE, 2020.

[CJLF16] Xu Chen, Lei Jiao, Wenzhong Li, and Xiaoming Fu. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Trans. Netw.*, 24(5):2795–2808, 2016.

[CLC22] Xing Chen, Jingtao Li, and Chaitali Chakrabarti. Energy and loss-aware selective updating for splitfed learning with energy harvesting-powered devices. *J. Signal Process. Syst.*, 94(10):961–975, 2022.

[CLM+21] Gilles Callebaut, Guus Leenders, Jarne Van Mulders, Geoffrey Ottoy, Lieven De Strycker, and Liesbet Van der Perre. The art of designing remote iot devices - technologies and strategies for a long battery life. *Sensors*, 21(3):913, 2021.

[CLY+24] Yang Cao, Shao-Yu Lien, Cheng-Hao Yeh, Der-Jiunn Deng, Ying-Chang Liang, and Dusit Niyato. Learning-based multitier split computing for efficient convergence of communication and computation. *IEEE Internet Things J.*, 11(20):33077–33096, 2024.

[CMGS20] Tejalal Choudhary, Vipul Kumar Mishra, Anurag Goswami, and Jagannathan Sarangapani. A comprehensive survey on model compression and acceleration. *Artif. Intell. Rev.*, 53(7):5113–5155, 2020.

[CR19] Jiasi Chen and Xukan Ran. Deep learning with edge computing: A review. *Proc. IEEE*, 107(8):1655–1674, 2019.

[CSB19] Chii Chang, Satish Narayana Srirama, and Rajkumar Buyya. Internet of things (iot) and new computing paradigms. In Rajkumar Buyya and Satish Narayana Srirama, editors, *Fog and Edge Computing*, Wiley Series on Parallel and Distributed Computing, pages 1–23. Wiley, 2019.

[CZ20] Shuo Cheng and Guohui Zhou. Facial expression recognition method based on improved VGG convolutional neural network. *Int. J. Pattern Recognit. Artif. Intell.*, 34(7):2056003:1–2056003:16, 2020.

[DBK+21] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[DBMC22] Pudi Dhilleswararao, Srinivas Boppu, M. Sabarimalai Manikandan, and Linga Reddy Cenkeramaddi. Efficient hardware architectures for accelerating deep neural networks: Survey. *IEEE Access*, 10:131788–131828, 2022.

[DJ14] Kalyanmoy Deb and Himanshu Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. *IEEE Trans. Evol. Comput.*, 18(4):577–601, 2014.

[DLH+20] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proc. IEEE*, 108(4):485–532, 2020.

[DPM+22] Aaron Yi Ding, Ella Peltonen, Tobias Meuser, Atakan Aral, Christian Becker, Schahram Dustdar, Thomas Hiessl, Dieter Kranzlmüller, Madhusanka Liyanage, Setareh Maghsudi, Nitinder Mohan, Jörg Ott, Jan S. Rellermeyer, Stefan Schulte, Henning Schulzrinne, Gürkan Solmaz, Sasu Tarkoma, Blesson Varghese, and Lars C. Wolf. Roadmap for edge AI: a dagstuhl perspective. *Comput. Commun. Rev.*, 52(1):28–33, 2022.

[DRL+23] Anurag Dutt, Sri Pramodh Rachuri, Ashley Lobo, Nazeer Shaik, Anshul Gandhi, and Zhenhua Liu. Evaluating the energy impact of device parameters for DNN inference on edge. In *Proceedings of the 14th International Green and Sustainable Computing Conference, IGSC 2023, Toronto, ON, Canada, October 28-29, 2023*, pages 52–55. ACM, 2023.

[DZF+20] Shuiguang Deng, Hailiang Zhao, Weijia Fang, Jianwei Yin, Schahram Dustdar, and Albert Y. Zomaya. Edge intelligence: The confluence of edge computing and artificial intelligence. *IEEE Internet Things J.*, 7(8):7457–7469, 2020.

[EAP21] Amir Erfan Eshratifar, Mohammad Saeed Abrishami, and Massoud Pedram. Jointdnn: An efficient training and inference engine for intelligent mobile cloud computing services. *IEEE Trans. Mob. Comput.*, 20(2):565–576, 2021.

[EEP19] Amir Erfan Eshratifar, Amirhossein Esmaili, and Massoud Pedram. Bottlenet: A deep learning architecture for intelligent mobile cloud computing services. In *2019 IEEE/ACM International Symposium on Low Power Electronics and Design, ISLPED 2019, Lausanne, Switzerland, July 29-31, 2019*, pages 1–6. IEEE, 2019.

[FDHM22] Seth G. Fitzgerald, Gary W. Delaney, David Howard, and Frédéric Maire. Evolving polydisperse soft robotic jamming grippers. In Jonathan E. Fieldsend and Markus Wagner, editors, *GECCO '22: Genetic and Evolutionary Computation Conference, Companion Volume, Boston, Massachusetts, USA, July 9 - 13, 2022*, pages 707–710. ACM, 2022.

[Fie17] Jonathan E. Fieldsend. University staff teaching allocation: formulating and optimising a many-objective problem. In Peter A. N. Bosman, editor, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, Berlin, Germany, July 15-19, 2017*, pages 1097–1104. ACM, 2017.

[GBC16] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016.

[GDDM14] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pages 580–587. IEEE Computer Society, 2014.

[GL87] Robert D. Galliers and Frank F. Land. Viewpoint: Choosing appropriate information systems research methodologies. *Commun. ACM*, 30(11):901–902, 11 1987.

[Goo24] Google. Tensorflow models on the edge tpu. `https://coral.ai/docs/edgetpu/models-intro/`, 2024. Accessed: 2024-11-22.

[HD24] Maruf Hassan and Steven Davy. Spikebottlenet: Spike-driven feature compression architecture for edge-cloud co-inference, 2024.

[HMD16] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[HMPR04] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Q.*, 28(1):75–105, 2004.

[HMS21] Walid A. Hanafy, Tergel Molom-Ochir, and Rohan Shenoy. Design considerations for energy-efficient inference on edge devices. In Herman de Meer

and Michela Meo, editors, *e-Energy '21: The Twelfth ACM International Conference on Future Energy Systems, Virtual Event, Torino, Italy, 28 June - 2 July, 2021*, pages 302–308. ACM, 2021.

[HPA+19] Andrew Howard, Ruoming Pang, Hartwig Adam, Quoc V. Le, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, and Yukun Zhu. Searching for mobilenetv3. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 1314–1324. IEEE, 2019.

[HTR22] Hanan Hussain, P. S. Tamizharasan, and C. S. Rahul. Design possibilities and challenges of DNN models: a review on the perspective of end devices. *Artif. Intell. Rev.*, 55(7):5109–5167, 2022.

[HVD15] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

[HZC+17] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

[IMA+16] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.

[INY21] Sohei Itahara, Takayuki Nishio, and Koji Yamamoto. Packet-loss-tolerant split inference for delay-sensitive deep learning in lossy wireless networks. In *IEEE Global Communications Conference, GLOBECOM 2021, Madrid, Spain, December 7-11, 2021*, pages 1–6. IEEE, 2021.

[JD14] Himanshu Jain and Kalyanmoy Deb. An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part II: handling constraints and extending to an adaptive approach. *IEEE Trans. Evol. Comput.*, 18(4):602–622, 2014.

[JJLM18] Hyuk-Jin Jeong, InChang Jeong, Hyeon-Jae Lee, and Soo-Mook Moon. Computation offloading for machine learning web apps in the edge server

environment. In *38th IEEE International Conference on Distributed Computing Systems, ICDCS 2018, Vienna, Austria, July 2-6, 2018*, pages 1492–1499. IEEE Computer Society, 2018.

[JKC+18] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 2704–2713. Computer Vision Foundation / IEEE Computer Society, 2018.

[KC07] Barbara Ann Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, 07 2007.

[KG19] Taranjit Kaur and Tapan Kumar Gandhi. Automated brain image classification based on VGG-16 and transfer learning. In *2019 International Conference on Information Technology (ICIT), Bhubaneswar, India, December 19-21, 2019*, pages 94–98. IEEE, 2019.

[KHG+17] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor N. Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. In Yunji Chen, Olivier Temam, and John Carter, editors, *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2017, Xi'an, China, April 8-12, 2017*, pages 615–629. ACM, 2017.

[KNH+22] Salman H. Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *ACM Comput. Surv.*, 54(10s):200:1–200:41, 2022.

[Kro08] Anders Krogh. What are artificial neural networks? *Nature biotechnology*, 26(2):195–197, 2008.

[LAB23] Daniel Luger, Atakan Aral, and Ivona Brandic. Cost-aware neural network splitting and dynamic rescheduling for edge intelligence. In Atakan Aral, editor, *Proceedings of the 6th International Workshop on Edge Systems, Analytics and Networking, EdgeSys 2023, Rome, Italy, 8 May 2023*, pages 42–47. ACM, 2023.

[LB24] Xian Li and Suzhi Bi. Optimal AI model splitting and resource allocation for device-edge co-inference in multi-user wireless sensing systems. *IEEE Trans. Wirel. Commun.*, 23(9):11094–11108, 2024.

[LFEF24] Hao Liu, Mohammed E. Fouda, Ahmed M. Eltawil, and Suhaib A. Fahmy. Split DNN inference for exploiting near-edge accelerators. In Rong N. Chang, Carl K. Chang, Jingwei Yang, Zhi Jin, Michael Sheng, Jing Fan, Kenneth Fletcher, Qiang He, Nimanthi L. Atukorala, Hongyue Wu, Shiqiang Wang, Shuiguang Deng, Nirmit Desai, Gopal Pingali, Javid Taheri, K. V. Subramaniam, Feras M. Awaysheh, Kaoutar El Maghaouri, and Yingjie Wang, editors, *IEEE International Conference on Edge Computing and Communications, EDGE 2024, Shenzhen, China, July 7-13, 2024*, pages 84–91. IEEE, 2024.

[LL20] Wenbin Li and Matthieu Liewig. A survey of AI accelerators for edge environment. In Álvaro Rocha, Hojjat Adeli, Luís Paulo Reis, Sandra Costanzo, Irena Orovic, and Fernando Moreira, editors, *Trends and Innovations in Information Systems and Technologies - Volume 2, WorldCIST 2020, Budva, Montenegro, 7-10 April 2020*, volume 1160 of *Advances in Intelligent Systems and Computing*, pages 35–44. Springer, 2020.

[LLW+18] Guangli Li, Lei Liu, Xueying Wang, Xiao Dong, Peng Zhao, and Xiaobing Feng. Auto-tuning neural network quantization framework for collaborative inference between the cloud and edge. In Vera Kurková, Yannis Manolopoulos, Barbara Hammer, Lazaros S. Iliadis, and Ilias Maglogiannis, editors, *Artificial Neural Networks and Machine Learning - ICANN 2018 - 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I*, volume 11139 of *Lecture Notes in Computer Science*, pages 402–411. Springer, 2018.

[LLW19] Zhongjie Lin, Hugh H. T. Liu, and Mike Wotton. Kalman filter-based large-scale wildfire monitoring with a system of uavs. *IEEE Trans. Ind. Electron.*, 66(1):606–615, 2019.

[LMAS23] Ibtissam Labriji, Mattia Merluzzi, Fatima Ezzahra Airod, and Emilio Calvanese Strinati. Energy-efficient cooperative inference via adaptive deep neural network splitting at the edge. In *IEEE International Conference on Communications, ICC 2023, Rome, Italy, May 28 - June 1, 2023*, pages 1712–1717. IEEE, 2023.

[LMP+21] Ivan Lujic, Vincenzo De Maio, Klaus Pollhammer, Ivan Bodrozic, Josip Lasic, and Ivona Brandic. Increasing traffic safety with real-time edge analytics and 5g. In Aaron Yi Ding and Richard Mortier, editors, *EdgeSys@EuroSys 2021: 4th International Workshop on Edge Systems, Analytics and Networking, Online Event, United Kingdom, April 26, 2021*, pages 19–24. ACM, 2021.

[LOD18] He Li, Kaoru Ota, and Mianxiong Dong. Learning iot in edge: Deep learning for the internet of things with edge computing. *IEEE Netw.*, 32(1):96–101, 2018.

[LVA+20] Stefanos Laskaridis, Stylianos I. Venieris, Mário Almeida, Ilias Leontiadis, and Nicholas D. Lane. SPINN: synergistic progressive inference of neural networks over device and cloud. In *MobiCom '20: The 26th Annual International Conference on Mobile Computing and Networking, London, United Kingdom, September 21-25, 2020*, pages 37:1–37:15. ACM, 2020.

[LWWW24] Zuguang Li, Wen Wu, Shaohua Wu, and Wei Wang. Adaptive split learning over energy-constrained wireless edge networks. In *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications Workshops, Vancouver, BC, Canada, May 20, 2024*, pages 1–6. IEEE, 2024.

[LZY+22] Jie Li, Laiping Zhao, Yanan Yang, Kunlin Zhan, and Keqiu Li. Tetris: Memory-efficient serverless inference through tensor sharing. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, Carlsbad, CA, July 2022. USENIX Association.

[MBCM22] Javier Mendez, Kay Bierzynski, Manuel P. Cuéllar, and Diego Pedro Morales. Edge intelligence: Concepts, architectures, applications, and future directions. *ACM Trans. Embed. Comput. Syst.*, 21(5):48:1–48:41, 2022.

[MCB+20] Yoshitomo Matsubara, Davide Callegaro, Sabur Baidya, Marco Levorato, and Sameer Singh. Head network distillation: Splitting distilled deep neural networks for resource-constrained edge computing systems. *IEEE Access*, 8:212177–212193, 2020.

[MCS+22] Yoshitomo Matsubara, Davide Callegaro, Sameer Singh, Marco Levorato, and Francesco Restuccia. Bottlefit: Learning compressed representations in deep neural networks for effective and efficient split computing. In *23rd IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, WoWMoM 2022, Belfast, United Kingdom, June 14-17, 2022*, pages 337–346. IEEE, 2022.

[MLR23] Yoshitomo Matsubara, Marco Levorato, and Francesco Restuccia. Split computing and early exiting for deep learning applications: Survey and research challenges. *ACM Comput. Surv.*, 55(5):90:1–90:30, 2023.

[MMR+17] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In Aarti Singh and Xiaojin (Jerry) Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 2017.

[MP43] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[MPCD19] Oskar Marko, Dejan Pavlovic, Vladimir S. Crnojevic, and Kalyanmoy Deb. Optimisation of crop configuration using NSGA-III with categorical genetic operators. In Manuel López-Ibáñez, Anne Auger, and Thomas Stützle, editors, *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*, pages 223–224. ACM, 2019.

[MTIB24] Daniel May, Alessandro Tundo, Shashikant Ilager, and Ivona Brandic. DynaSplit: A Hardware-Software Co-Design Framework for Energy-Aware Inference on Edge, 2024.

[MVO+24] Akrit Mudvari, Antero Vainio, Iason Ofeidis, Sasu Tarkoma, and Leandros Tassiulas. Adaptive compression-aware split learning and inference for enhanced network efficiency. *ACM Trans. Internet Technol.*, 24(4), November 2024.

[NZES05] Patrick Ngatchou, Anahita Zarei, and A El-Sharkawi. Pareto multi objective optimization. In *Proceedings of the 13th international conference on, intelligent systems application to power systems*, pages 84–91. IEEE, 2005.

[PCMP21] Daniele Jahier Pagliari, Roberta Chiaro, Enrico Macii, and Massimo Poncino. CRIME: input-dependent collaborative inference for recurrent neural networks. *IEEE Trans. Computers*, 70(10):1626–1639, 2021.

[PCT+20] Edward B Panganiban, Wen-Yaw Chung, Wei-Chieh Tai, Arnold C Paglinawan, Jheng-Siang Lai, Ren-Wei Cheng, Ming-Kai Chang, and Po-Hsuan Chang. Real-time intelligent healthcare monitoring and diagnosis system through deep learning and segmented analysis. In *Future Trends in Biomedical and Health Informatics and Cybersecurity in Medical Devices: Proceedings of the International Conference on Biomedical and Health Informatics, ICBHI 2019, 17-20 April 2019, Taipei, Taiwan*, pages 15–25. Springer, 2020.

[QLD+24] Danfeng Qin, Chas Leichner, Manolis Delakis, Marco Fornoni, Shixin Luo, Fan Yang, Weijun Wang, Colby R. Banbury, Chengxi Ye, Berkin Akin, Vaibhav Aggarwal, Tenghui Zhu, Daniele Moro, and Andrew Howard. Mobilenetv4 - universal models for the mobile ecosystem. *CoRR*, abs/2404.10518, 2024.

[RDGF16] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 779–788. IEEE Computer Society, 2016.

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual

Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[RF17] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 6517–6525. IEEE Computer Society, 2017.

[RF18] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[RMEZ23] Brendan C Reidy, Mohammadreza Mohammadi, Mohammed E Elbtity, and Ramtin Zand. Efficient deployment of transformer models on edge TPU accelerators: A real system evaluation. In *Architecture and System Support for Transformer Models (ASSYST @ISCA 2023)*, 2023.

[RN20] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020.

[Sat17] Mahadev Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.

[Sch15] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[SCZ+16] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet Things J.*, 3(5):637–646, 2016.

[SDSE20] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green AI. *Commun. ACM*, 63(12):54–63, 2020.

[SGM19] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 3645–3650. Association for Computational Linguistics, 2019.

[SGSB+15] Pablo Serrano, Andres Garcia-Saavedra, Giuseppe Bianchi, Albert Banchs, and Arturo Azcorra. Per-frame energy consumption in 802.11 devices and its implication on modeling and design. *IEEE/ACM Transactions on Networking*, 23(4):1243–1256, 2015.

[SHZ+18] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition,*

CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018, pages 4510–4520. Computer Vision Foundation / IEEE Computer Society, 2018.

[SMB22] Eric Samikwa, Antonio Di Maio, and Torsten Braun. ARES: adaptive resource-aware split learning for internet of things. *Comput. Networks*, 218:109380, 2022.

[SRG24] SRG Research. Cloud Market Growth Surge Continues in Q3: Growth Rate Increases for the Fourth Consecutive Quarter, 2024. Accessed: 2024-11-22.

[SSWP21] Daniel Schwartz, Jonathan Michael Gomes Selman, Peter H. Wrege, and Andreas Paepcke. Deployment of embedded edge-ai for wildlife monitoring in remote regions. In M. Arif Wani, Ishwar K. Sethi, Weisong Shi, Guangzhi Qu, Daniela Stan Raicu, and Ruoming Jin, editors, *20th IEEE International Conference on Machine Learning and Applications, ICMLA 2021, Pasadena, CA, USA, December 13-16, 2021*, pages 1035–1042. IEEE, 2021.

[SZ15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[TL19] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 2019.

[TL21] Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 10096–10106. PMLR, 2021.

[TMI+23] Alessandro Tundo, Marco Mobilio, Shashikant Ilager, Ivona Brandic, Ezio Bartocci, and Leonardo Mariani. An energy-aware approach to design self-adaptive ai-based applications on the edge. In *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023, Luxembourg, September 11-15, 2023*, pages 281–293. IEEE, 2023.

[TWWC19] Zhenheng Tang, Yuxin Wang, Qiang Wang, and Xiaowen Chu. The impact of GPU DVFS on the energy and performance of deep learning: an empirical study. In *Proceedings of the Tenth ACM International Conference on Future Energy Systems, e-Energy 2019, Phoenix, AZ, USA, June 25-28, 2019*, pages 315–325. ACM, 2019.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[WHL+20] Xiaofei Wang, Yiwen Han, Victor C. M. Leung, Dusit Niyato, Xueqiang Yan, and Xu Chen. *Edge AI - Convergence of Edge Computing and Artificial Intelligence*. Springer, 2020.

[WZWY22] Xudong Wang, Li Lyna Zhang, Yang Wang, and Mao Yang. Towards efficient vision transformer inference: a first study of transformers on mobile devices. In Robert LiKamWa and Urs Hengartner, editors, *HotMobile '22: The 23rd International Workshop on Mobile Computing Systems and Applications, Tempe, Arizona, USA, March 9 - 10, 2022*, pages 1–7. ACM, 2022.

[XXW+23] Yilin Xiao, Liang Xiao, Kunpeng Wan, Helin Yang, Yi Zhang, Yi Wu, and Yanyong Zhang. Reinforcement learning based energy-efficient collaborative inference for mobile edge computing. *IEEE Trans. Commun.*, 71(2):864–876, 2023.

[YHC+18] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In Yike Guo and Faisal Farooq, editors, *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pages 974–983. ACM, 2018.

[YHPC18] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing [review article]. *IEEE Comput. Intell. Mag.*, 13(3):55–75, 2018.

[YLH+18] Wei Yu, Fan Liang, Xiaofei He, William G. Hatcher, Chao Lu, Jie Lin, and Xinyu Yang. A survey on the edge computing for the internet of things. *IEEE Access*, 6:6900–6919, 2018.

[YLL+20] Shuochao Yao, Jinyang Li, Dongxin Liu, Tianshi Wang, Shengzhong Liu, Huajie Shao, and Tarek F. Abdelzaher. Deep compressive offloading: speeding up neural network inference by trading edge computation for network latency. In Jin Nakazawa and Polly Huang, editors, *SenSys '20: The 18th ACM Conference on Embedded Networked Sensor Systems, Virtual Event, Japan, November 16-19, 2020*, pages 476–488. ACM, 2020.

[YW23] Ya-Ting Yang and Hung-Yu Wei. A coalition formation approach for privacy and energy-aware split deep learning inference in edge camera network. *IEEE Trans. Netw. Serv. Manag.*, 20(3):3673–3685, 2023.

[ZCX21] Letian Zhang, Lixing Chen, and Jie Xu. Autodidactic neurosurgeon: Collaborative deep inference for mobile edge intelligence via online learning. In Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia, editors, *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 3111–3123. ACM / IW3C2, 2021.

[ZLL+20] Shigeng Zhang, Yinggang Li, Xuan Liu, Song Guo, Weiping Wang, Jianxin Wang, Bo Ding, and Di Wu. Towards real-time cooperative deep inference over the cloud and edge end devices. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 4(2):69:1–69:24, 2020.

[ZMB+21] Daniel Zhang, Saurabh Mishra, Erik Brynjolfsson, John Etchemendy, Deep Ganguli, Barbara J. Grosz, Terah Lyons, James Manyika, Juan Carlos Niebles, Michael Sellitto, Yoav Shoham, Jack Clark, and C. Raymond Perrault. The AI index 2021 annual report. *CoRR*, abs/2103.06312, 2021.

[ZSL+24] Chenxi Zhao, Min Sheng, Junyu Liu, Tianshu Chu, and Jiandong Li. Energy-efficient power control for multiple-task split inference in uavs: A tiny learning-based approach. *IEEE Internet Things J.*, 11(12):21146–21157, 2024.

[ZWZ+23] Hongxia Zhang, Dengyue Wang, Wei Zhang, Lizhuang Tan, Godfrey Kibalya, Peiying Zhang, and Kostromitin Konstantin Igorevich. Qos prediction in intelligent edge computing based on feature learning. *J. Cloud Comput.*, 12(1):17, 2023.

[ZZL+24] Ziyang Zhang, Yang Zhao, Huan Li, Changyao Lin, and Jie Liu. DVFO: learning-based DVFS for energy-efficient edge-cloud collaborative inference. *IEEE Trans. Mob. Comput.*, 23(10):9042–9059, 2024.