TU WIEN Informatics

Collaborative Inference for Edge Intelligence - Impacts on Performance and Privacy of Partition Points

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Lukas Pichler, BSc
Matrikelnummer 01526493

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dr. Schahram Dustdar
Mitwirkung: Dipl.-Ing. Clemens Lachner

Wien, 22. März 2022

Lukas Pichler	Schahram Dustdar

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

TU WIEN Informatics

Collaborative Inference for Edge Intelligence - Impacts on Performance and Privacy of Partition Points

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Lukas Pichler, BSc
Registration Number 01526493

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dr. Schahram Dustdar
Assistance: Dipl.-Ing. Clemens Lachner

Vienna, 22nd March, 2022

Lukas Pichler	Schahram Dustdar

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Lukas Pichler, 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.

Wien, 22. März 2022

Lukas Pichler

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Danksagung

An dieser Stelle möchte ich mich herzlich bei meinem Betreuer Schahram Dustdar und Mitbetreuer Clemens Lachner bedanken. Danke für das ständige Feedback und die Diskussionen, die mir sehr dabei geholfen haben diese Arbeit zu vollenden.

Weiters möchte ich ein großes Dankeschön an meine Freunde und Familie aussprechen, die mich seit dem Anfang des Studiums stets unterstützt haben. Ohne euch hätte ich es nie geschafft dieses zu absolvieren.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

At this point I want to express my gratitude towards my advisor Schahram Dustdar and co-advisor Clemens Lachner. Thank you for the continuous feedback and input, which has helped me tremendously to finish this thesis.

I also want to thank my friends and family, who have supported me throughout the whole duration of my studies. I would have never been able to complete them without you.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Edge Intelligenz kombiniert die Prinzipien von Edge Computing (EC) und Künstlicher Intelligenz (KI) um die Ausführung von KI-basierten Applikationen an den Rand des Netzwerks zu bringen. Jedoch wirft die heterogene Infrastruktur in EC Herausforderungen für die Performance und Privatsphäre solcher Applikationen auf, wie zum Beispiel Video Analysis Pipelines (VAPs). VAPs verwenden die neuesten Machine Learning (ML) Modelle, um Videostreams zu analysieren, aber werden durch die geringe Leistung der Geräte am Rand des Netzwerks limitiert. Zusätzlich verarbeiten VAPs potenziell auch personenbezogene Daten. Kollaborative Inferenz (Ko-Inferenz) ist eine Technik, um die Bedenken bezüglich Performance und Privatsphäre zu mindern, indem der Rechenaufwand der Inferenz aufgeteilt wird. Konkret wird dabei ein ML Modell an einem Partitionspunkt geteilt, wodurch zwei Modelle entstehen, die auf verschiedenen Geräten verteilt werden. Nachdem ein Gerät den ersten Teil des ML Modells ausführt, werden die Zwischenergebnisse (ZE) an ein anderes Gerät übertragen. Dieses vollendet die Inferenz mit der zweiten Hälfte des ML Modells und sendet das Endergebnis zurück. Dieser Ansatz kann eine effizientere Ausführung der Analyse ermöglichen und gleichzeitig Datenschutzaspekte verbessern, da statt rohen (Video) Daten ZE übertragen und verarbeitet werden. Das Bestimmen eines optimalen Partitionspunkt ist für die Ko-Inferenz von größter Bedeutung, um die Performance und Privatsphäre einer VAP zu maximieren.

Diese Arbeit analysiert die Auswirkung verschiedener Partitionspunkte auf die Performance und Privatsphäre von Ko-Inferenz Systemen. Um die Auswirkung auf Performance möglichst realitätsgetreu zu evaluieren, wurde eine prototypische Ko-Inferenz VAP für Objekterkennung auf einer für EC typischen heterogenen Hardware-Testumgebung umgesetzt. Zusätzlich wurde eine Rekonstruktionsattacke auf die verschiedenen Partitionspunkte angewandt, um die Rekonstruierbarkeit von Originalbildern mit ZE und den damit verbundenen Sicherheitsaspekten zu erforschen.

Die resultierenden Performance-Benchmarks haben gezeigt wie die verschiedenen Partitionspunkte, in Summe neun, der implementierten Ko-Inferenz VAP den Durchsatz, Utilisierung von Ressourcen, und benötigter Bandbreite für die verschiedenen Geräte der Testumgebung beeinflussen. Die Rekonstruktionsattacke generierte für sechs Partitionspunkte Bilder mit personenbezogenen Informationen. Die generierten Bilder für die restlichen drei Partitionspunkte beinhalteten keine solcher Informationen, wodurch die Ko-Inferenz VAP gegen die angewandte Rekonstruktionsattacke geschützt ist.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

Edge computing combined with Artificial Intelligence allows for a powerful new paradigm called Edge Intelligence, pushing the execution of AI-based applications towards the edge of the network. However, the heterogeneous infrastructure of edge computing poses challenges to performance and privacy of these applications, such as Video Analysis Pipelines (VAPs). VAPs utilize state-of-the-art machine learning models to analyze video streams, but their execution is limited at the edge, due to the insufficient performance characteristics of edge devices. Furthermore, they process data potentially containing personally identifiable information. Collaborative Inference (Co-Inference) poses a novel technique to alleviate both performance and privacy concerns, by distributing the computational workload of inference. Concretely, machine learning models are split at a partition point, resulting in a head and tail model, which are then deployed on different compute nodes. After performing inference with the head model on one device, resulting intermediate features are transmitted to another device, which finishes inference with the tail model. This collaborative approach can accelerate execution of the analysis, and improve data protection aspects, as intermediate features are transmitted and processed instead of raw (video) data. Determining an optimal partition point for Co-Inference is of paramount importance in order to maximize performance and privacy aspects of a VAP.

This thesis analyzes the impact different partition points have on the performance and privacy of Co-Inference systems. To evaluate the impact on performance as close to real life scenarios as possible, a prototypical Co-Inference VAP for object detection was implemented on a heterogeneous hardware test environment typical for edge computing. Additionally, an image reconstruction attack was performed on the different partition points, to evaluate the reconstructability of original images from intermediate features and related privacy concerns.

The performance benchmarks have shown how the different partition points, nine in total, of the implemented Co-Inference VAP impact throughput, resource utilization, and required bandwidth for the various devices of the testbed. Images exposing personally identifiable information were generated when applying the image reconstruction attack to six partition points. The generated images for the remaining three partition points did not expose such information, therefore protecting the Co-Inference VAP against the applied image reconstruction attack.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Kurzfassung **xi**

Abstract **xiii**

Contents **xv**

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Problem Statement . . . 2
1.3 Research Questions . . . 2
1.4 Research Methods . . . 3
1.5 Structure . . . 4

2 Fundamentals **5**
2.1 Edge Computing . . . 5
2.2 Artificial Intelligence . . . 6
2.3 Engineering Edge Intelligence . . . 11
2.4 Data Protection and Personally Identifiable Information . . . 16

3 Impact on Performance and Privacy of Partition Points in a Prototypical Co-Inference VAP **19**
3.1 Overview . . . 19
3.2 Video Analysis Pipeline Setup . . . 20
3.3 Model Partition . . . 22
3.4 Quantization . . . 25
3.5 Image Reconstruction Attack . . . 25

4 Methodology **29**
4.1 Research Questions . . . 29
4.2 Evaluation Environment . . . 29
4.3 Benchmarking Performance . . . 30
4.4 Reconstructability Benchmarks . . . 34

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

5 Results **35**
5.1 Performance Benchmarks 35
5.2 Image Reconstruction Attack 50

6 Discussion and Related Work **59**
6.1 Performance Benchmarks 59
6.2 Image Reconstruction Attack and Impact on Data Protection 67
6.3 Summary of the Key Findings 72

7 Conclusion **75**
7.1 Domain, Motivation, and Problem Statement 75
7.2 Research Questions 76
7.3 Future Work 78

A Remaining Benchmarks **79**
A.1 Co-Inference VAP Performance Benchmarks 79
A.2 Comparison of Local and Collaborative Benchmarks 83
A.3 Resource Utilization of Model Types 84

List of Figures **87**

List of Tables **91**

Bibliography **93**

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 1

Introduction

1.1 Motivation

Edge computing combined with Artificial Intelligence allows for a powerful new paradigm called Edge Intelligence, which aims to facilitate highly performant AI-assisted applications at the edge of the network. This could enable countless scenarios, as facial recognition, object tracing, vehicle detection and more, can be used for smart city applications requiring highly distributed AI-based systems, e.g., detecting traffic accidents and notifying emergency services.

These applications often leverage Convolutional Neural Networks (CNNs) in state-of-the-art machine learning models. Through their versatility and recent advances in respective research, CNNs are widespread among computer vision applications, such as Video Analysis Pipelines (VAPs). These pipelines can process video streams from multiple cameras with the help of AI, to generate alarms or act on what has been detected on video. Analyzing videos in VAPs, is often facilitated by offloading video streams to powerful cloud servers, where computational heavy tasks of inference are performed. The quality of service this approach provides varies greatly due to the high bandwidth usage and latency within the network, which edge computing promises to alleviate, by pushing computation towards the edge. Additionally, this approach entails privacy concerns, as the receiving server could process the raw video data unlawfully, violating the privacy of everyone visible on video. Concerns regarding privacy have emerged especially in the public sector, as AI-assisted computer vision applications continue to permeate deeper into our daily lives. Furthermore, the General Data Protection Regulation (GDPR) [Cou22] mandates the protection of personally identifying information, which necessitates the development of privacy protecting systems.

With the improvement of machine learning models for computer vision applications, their size and computational overhead decreases. This allows the execution of inference

tasks on resource constrained devices at the edge of the network, instead of being bound to a cloud environment exclusively. Current state-of-the-art solutions, which can be run solely on edge devices, mitigate threats to privacy to some extent, as they could process data locally without the need of transmission. However, the inferior performance characteristics of typical edge devices diminish their usability, resulting in high research interest for solutions.

1.2 Problem Statement

A promising strategy to improve performance on the edge is Collaborative Inference (Co-Inference), as proposed by [TMK17, KHG+17, LZC18], where the computational workload of inference is distributed. For this approach the used machine learning model is split up at a partition point, resulting in two parts of the original model, commonly referred to as head and tail model. An edge device performs inference with the head model, producing intermediate features. These are then transmitted to a server which finishes inference with the tail model and returns the results.

As this method exchanges intermediate features instead of raw video data, receiving servers with malicious intent can not immediately view the original data, as intermediate features cannot be easily reverted. Specific image reconstruction attacks are needed to gather potentially sensitive information from the source video data. There are several approaches, such as [MV15, DB16, OL19, HZL19], that aim to reconstruct original images from their intermediate features, which once again pose a threat to data protection in Co-Inference solutions.

The performance improvement and data protection aspects of intermediate features in a Co-Inference Video Analysis Pipeline (VAP) depend on the selected partition point. Therefore, determining an optimal partition point in Co-Inference is of paramount importance in order to maximize performance and privacy aspects of a VAP. Hence, this thesis deals with the question of how different partition points of a Co-Inference VAP impact both performance and privacy.

1.3 Research Questions

As both performance and privacy of VAPs are of great research interest, this work aims to answer the following research questions.

- **RQ$_1$**: How do the different possible partition points of a Co-Inference VAP impact the performance in regard to throughput, resource utilization, and required bandwidth?

  Co-Inference promises to improve performance for resource constrained devices, as presented in [KHG+17, LZC18]. This thesis aims to evaluate how different partition points of a Co-Inference VAP affect the various performance aspects of

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

the research question. Choosing an adequate partition point in future Co-Inference systems should be eased by the results of this thesis. Results should further show if Co-Inference can push the computation related with VAPs further towards the edge, possibly allowing real time analysis without the need of typical offloading.

- **RQ$_2$**: How do the different possible partition points of a Co-Inference VAP impact security and data protection?

  Typical offloading entails that the receiving server is able to process the raw data. Even if the transmission is encrypted, decryption is necessary on the server to perform inference. This causes privacy concerns, as the GDPR [Cou22] mandates that personal data has to be protected from unlawful processing, which a server with malicious intent could easily accomplish. In the case of Co-Inference, intermediate features are transmitted instead, possibly protecting the data against malicious adversaries. However, multiple approaches, which aim to reconstruct original images from intermediate features, pose a threat to the privacy in Co-Inference systems as well. The success of these approaches depends on the chosen partition point. Therefore, as many partition points as possible are evaluated for the given VAP.

1.4 Research Methods

The goal of this work is two-fold. First, the impact of partition point on performance in Co-Inference is evaluated. Second, as personally identifiable information has to be protected to comply with the GDPR, the impact of partition point on privacy and data protection in Co-Inference is analyzed as well.

In this thesis a Co-Inference VAP is implemented, using a state-of-the-art machine learning model for object detection, in order to analyze various performance aspects, such as throughput, resource utilization, and required bandwidth. The used machine learning model is further partitioned at multiple points, to show how each of the resulting partition points impact aforementioned performance aspects. To make well-founded statements about this impact, the Co-Inference VAP and its available partition points are evaluated on multiple test devices. This is done due to the heterogeneous nature in system composition of edge computing, where edge devices may be able to outperform a Co-Inference system.

Potential privacy protecting capabilities enabled by Co-Inference are evaluated. An image reconstruction attack is used to determine if transmission of intermediate features can protect the original image data. This type of attack tries to generate the original image from intermediate features. All possible partition points of the Co-Inference VAP are subjected to this attack. To objectively quantify the similarity of generated and original images, three metrics are calculated for the results of each partition point.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

1.5 Structure

The structure of this thesis is as follows. In Chapter 2, different domains important to this thesis and their fundamentals are explained. Chapter 3 describes the setup of the experiments that were performed within this thesis. In Chapter 4, an overview of the methodological approach carried out to answer the research questions is given. Results of the experiments can be found in Chapter 5, which are discussed and compared to related work in Chapter 6. Finally, a conclusion and future outlook on Co-Inference in the Edge Intelligence domain is given in Chapter 7.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 2

Fundamentals

This chapter deals with the fundamentals of each domain relevant to this thesis. A brief overview and state-of-the-art research is presented for each domain, including current challenges and limitations.

2.1 Edge Computing

Network latency, available bandwidth, security and privacy, availability and energy consumption were identified as the major problems of cloud computing [SPX19]. This results in a surge in interest for the new edge computing paradigm. Edge computing is defined by Shi and Dustdar [SD16] as *"the enabling technologies that allow computation to be performed at the network edge so that computing happens near data sources"*. An edge device itself is defined as *"any computing and network resources along the path between data sources and cloud data centers"* [SD16]. A graphical overview on edge computing can be found in Figure 2.1. Even more necessitated by the emergence of the Internet of Things; which entails a vast amount of data generated by interconnected edge devices; computation, storage and communication services are moved from the cloud to the edge of the network [VF13, p. 8]. The International Data Corporation predicts that there will be 55.7 billion devices, with 75% being connected to an Internet of Things platform, by 2025, which are estimated to generate a total of 73.1 Zettabytes of data [Int21].

Pushing computation towards the edge does not mitigate all problems the cloud computing paradigm faces. Still, programmability, naming, privacy and security, data abstraction, service management, and optimization metrics were identified as the remaining challenges of edge computing [SCZ+16]. Various edge computing applications, include video analysis, smart homes, smart cities and more.

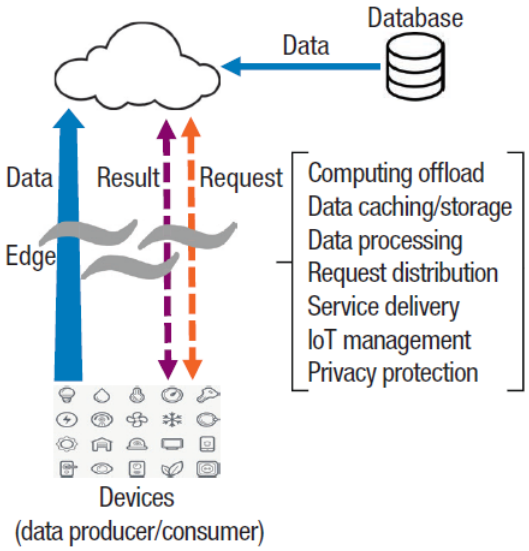

Figure 2.1: Edge computing overview, Shi and Dustdar [SD16] - Edge devices can generate and consume data, with nodes along the edge taking on some computational workload, e.g., data caching, data processing, with the end goal of storing data in the cloud

2.2 Artificial Intelligence

Artificial Intelligence is a broad field which entails different definitions on what exactly AI is. Russel and Norvig [RN02, p. 1] define four different versions of AI, which have been researched in the past. They are the possible combinations of two dimensions - human versus rational, thought versus behavior - which require their own method to be evaluated: Acting humanly, thinking humanly, acting rationally, and thinking rationally. One popular method to evaluate if a machine is able to act humanly is the Turing test, invented by Alan Turing in 1950 [TUR50], in which the goal of a computer is to trick a human interrogator into thinking that the computer is a human being.

Recent interest in AI has grown over the past years, which can be linked to the advancements made in various research topics, e.g., neural networks, natural language processing and computer vision. Additionally, advancements in non-AI related topics, e.g., bandwidth support through infrastructure, specialized hardware, and development frameworks continue to push boundaries in the field of what is possible with AI. Nowadays, AI encompasses multiple different subfields and techniques, each with its own research dedicated to it, e.g., natural language processing, machine learning, computer vision, which allows for a vast amount of different applications, such as self-driving cars, home assistants, or diagnosis assistance in the medical field.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

2.2.1 Machine Learning

Machine learning, which can be seen as a subfield of AI, has been defined by Russel and Norvig as when *"a computer observes some data, builds a model based on the data, and uses the model as both a hypothesis about the world and a piece of software that can solve problems"* [RN02, p. 651]. This can be applied to a wide variety of topics according to them, e.g., a piece of software which predicts the stock market or a robot which has to maneuver through a maze [RN02, p. 651]. In both instances, the computer program would have to learn new behavior, by either adjusting the predicted price according to new data, or navigating through an unknown maze.

A popular form of learning such behavior is called supervised learning, in which *"the agent observes input-output pairs and learns a function that maps from input to output"* [RN02, p. 651]. Often used for generating models for regression or classification problems, a labeled training set is needed, to learn a correct model, a function. The most basic classification example would be an email spam detector, which learns to mark emails as spam (or not) based on a training dataset, containing spam and non-spam mails marked as such. In this case, the training dataset would consist of pairs with the form *(mail, boolean)*, where the boolean signifies if the mail is a spam mail or not. This training data is of utmost importance, as the machine learning model can only learn the correct behavior, if the input-output pairs of the dataset are labeled correctly. After training, the model could be able to correctly label new, previously unseen, mails as spam or not spam, if trained correctly.

2.2.2 Deep Learning

Deep learning refines the definition of machine learning further, *"in which hypotheses take the form of complex algebraic circuits with tuneable connection strengths"*, according to Russel and Norvig [RN02, p. 750]. More commonly known as neural networks, these algebraic circuits stem from the works of McCulloch and Pitts, which came up with a logical calculus to model the nervous system as a network in 1943 [MP43]. These neural networks usually consist of multiple layers connected with each other, which is where the term "deep" in deep learning comes from [RN02, p. 750]. Popular applications of deep learning include, but are not limited to, computer vision tasks, such as object detection or face recognition.

Neural networks and their structure are very important when considering the bigger scope of this thesis, especially for partitioning of a neural network, which is why the most common architectures and building blocks of neural networks will be shortly presented.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

2.2.3 Neurons

Neural networks are made up of multiple *neurons* connected to each other. As can be seen in Figure 2.2, one neuron receives a variable amount j of input values $x_1, \ldots x_j$, a weight for each input value $w_1, \ldots w_j$, has an overall bias b and an activation function f.

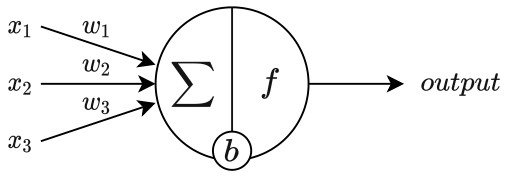

Figure 2.2: Neuron, adapted from [Nie15]

A sum is built over the products of each input value x_j and its respective weight w_j after which the bias is added to the sum, which can be seen in Formula 2.1.

$$z = \sum_j x_j w_j + b \tag{2.1}$$

The activation function, which differs based on what type of neuron is used, is then applied on the result z of above Formula 2.1. Up next is a non-exhaustive list of the most common neurons with their activation function.

Sigmoid Neurons

The activation function f of a sigmoid neuron is the following [Nie15]:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.2}$$

Its generates values between 0 and 1.

ReLU

In contrast to sigmoid neurons, ReLU neurons, which stands for Rectified Linear Unit, can generate values greater than 1, but do not propagate negative values. The output of one neuron is calculated with [Nie15]:

$$R(z) = max(0, z) \tag{2.3}$$

LeakyReLU

Instead of propagating 0 for negative values, LeakyReLU attempts to improve ReLU, by allowing negative values to *leak* to the next neuron with the following activation function, with a typically taking on the value of 0.01:

$$LR(z) = max(a * z, z) \tag{2.4}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

2.2.4 Neural Networks

A neural network typically consists of an input layer, zero to many hidden layers, and one output layer. Figure 2.3 shows a *fully connected network*, as every neuron of a layer is connected to each neuron of the next layer.

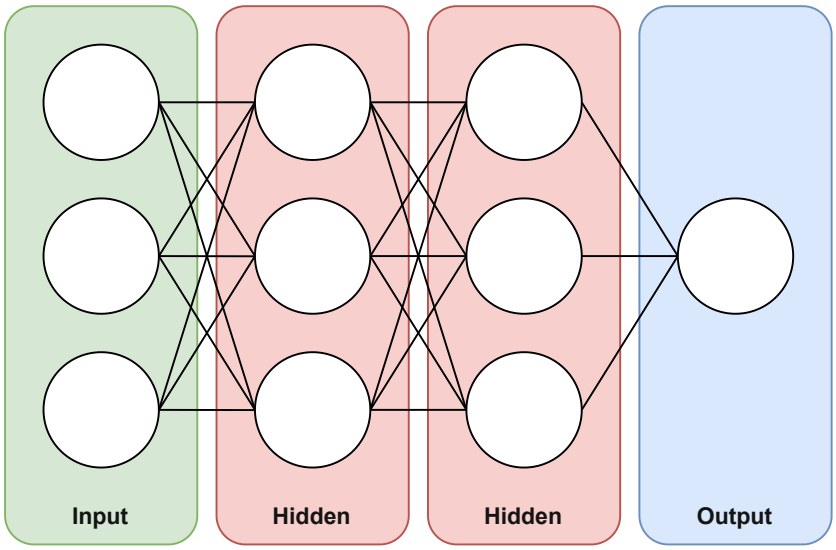

Figure 2.3: Fully connected neural network

Training

To calculate the values for weights and biases of the neurons within the network, the machine learning model has to be trained on multiple inputs. In case of object detection, or similar computer vision tasks, supervised learning, which was explained in Section 2.2.1, is usually used to train models. Over several iterations, loss is calculated based on the output of the network and the label values, which is used to shift the values of weights and biases in the correct direction. The backpropagation algorithm responsible for this is out of scope for this thesis, as it focuses on inference.

2.2.5 Convolutional Neural Networks

Key to many computer vision tasks, are Convolutional Neural Networks (CNNs), which rely on special layers, called *convolutional* layers.

Convolutional Layer

A convolutional layer slides one or multiple kernels over the original image, starting from the top left until it reaches the bottom right corner. These kernels have a specific size (in most cases of form $n*n$) and a stride length. This is also shown in Figure 2.4, where the red area symbolizes a kernel of 3x3 size with a stride length of 1, moving one pixel to

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

the right, or bottom when it has reached the end of the matrix, each time. The result of the matrix multiplication of the kernel and the section of the original image is passed through an activation function, which could be any of those in Section 2.2.3. The result of the activation function is symbolized by the green pixel in Figure 2.4. At the end of each kernel pass, a feature map is generated, which helps in detecting features, such as edges, textures, or patterns. Multiple kernels can be applied within a convolutional layer, allowing multiple feature maps to be generated from one layer, which increases the third dimension of the output.

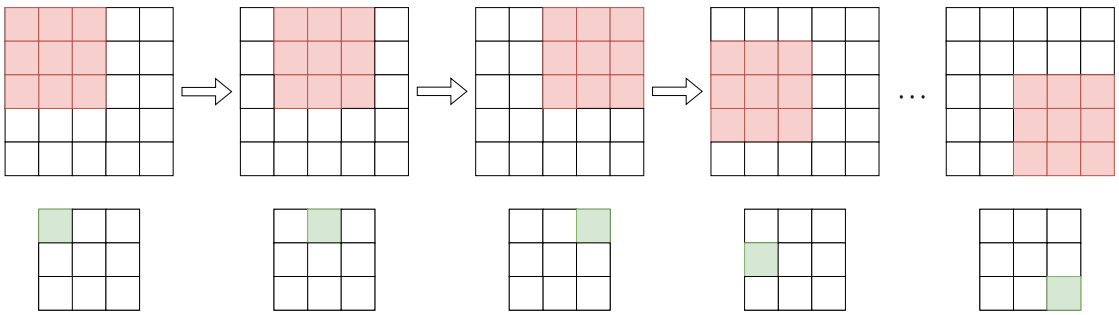

Figure 2.4: 2D Kernel with size 3, stride length 1, sliding across a 5x5 image, adapted from [Nie15]

Pooling Layer

Pooling layers work similar to convolutional layers, as in that they also slide an $n*n$ matrix across their input. Popular pooling variants include *max pooling*, which returns the maximum value, while *average pooling* returns the average of the $n*n$ values of the original section. Their stride length is usually the same as n, as one of the main goals of pooling is to reduce size. Therefore, the output of a pooling layer will have a smaller size than its input. This is also symbolized in Figure 2.5.

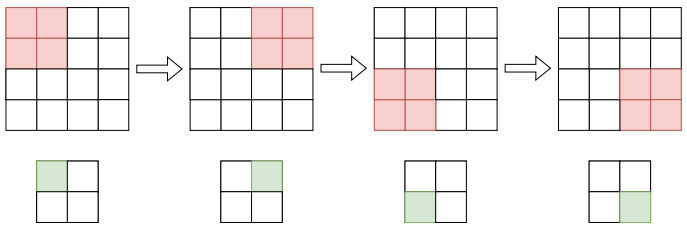

Figure 2.5: Pooling with filter size 2, stride length 2, sliding across a 4x4 matrix, adapted from [Nie15]

After generating feature maps with the repetition of convolutional and pooling layers, a fully connected network, as depicted in Figure 2.3, is often used to learn classification of these features. A resulting CNN may have a similar shape to that of Figure 2.6, which shows the architecture of a popular CNN for object detection, YOLO [RDGF16].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

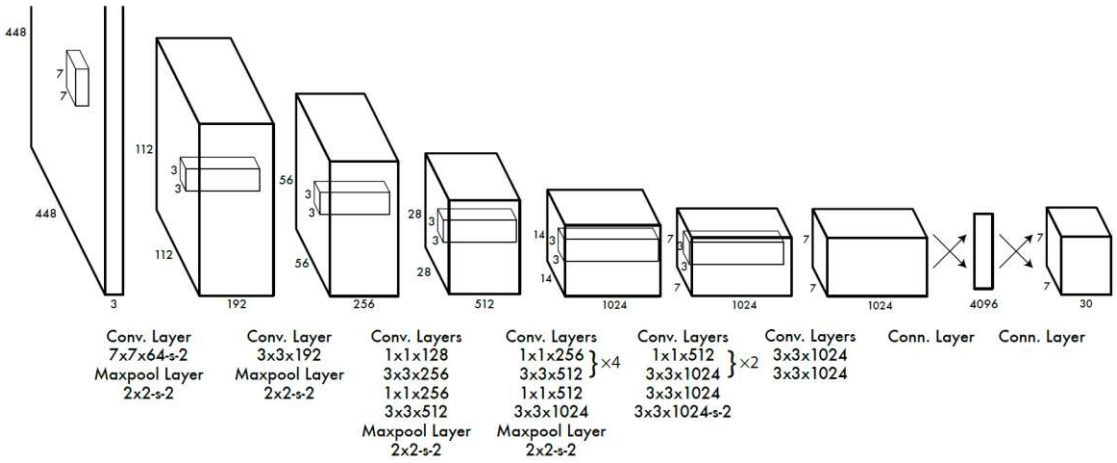

Figure 2.6: Architecture of YOLO, a CNN for object detection [RDGF16]

2.3 Engineering Edge Intelligence

This section presents an overview of what Edge Intelligence is, and how it can be achieved with state-of-the-art methods.

2.3.1 Edge Intelligence

Edge Intelligence has been defined as *"the marriage of edge computing and AI"* by Zhou et al. [ZCL+19]. While edge computing aims to process data at the edge of the network, Artificial Intelligence aims to imitate or extend human abilities. According to Zhou et al. [ZCL+19], combining these technologies is explained by their clear overlap: With the increase of data generated at the edge of the network through the Internet of Things, it is only natural to learn or perform inference, to generate more data or act on it, on the edge.

Deng et al. [DZF+20] divide the research of Edge Intelligence into two distinct groups, namely *AI for edge* and *AI on edge*. While the former focuses on utilizing AI to resolve constrained optimization problems, also defined as *intelligence-enabled edge computing*, the latter analyzes how AI models can be trained and executed on the edge of the network, with regard to performance, privacy, reliability, and more. *AI on edge* furthermore splits its research into Model Adaptation, Framework Design and Processor Acceleration.

Rausch and Dustdar [RD19] describe the evolution of the cyber-human, in which Edge Intelligence is the enabler to augment the human perception through various AI-based applications at the edge of the network. Currently available examples include Google Lens[1], which utilizes AI-based visual analysis in a smartphone application, to scan and translate text, identify plants or animals.

[1] https://lens.google/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

2.3.2 Development of AI-based Applications

Frameworks such as *TensorFlow*[2] and *PyTorch*[3] have greatly simplified development of machine learning models in recent years. Through extensive APIs, open-source communities, multi-language support, and more, it has never been easier to develop AI-based applications. To facilitate execution of these applications on the edge, these frameworks have been further developed to specifically alleviate pains of edge devices, including mobile phones. *TensorFlow Lite* is an optimized version for edge devices, as it addresses 5 key constraints of edge computing, according to Google[4]: latency, privacy, connectivity, size and power consumption. It furthermore includes support for hardware acceleration and model optimization. Conversion of TensorFlow models to TensorFlow Lite can be done via the included converter, but problems may arise due to limitations in operator compatibility, which has to be considered during machine learning model development.

2.3.3 Dedicated Hardware

Performing inference on CPUs will typically perform worse than on GPUs. This is due to GPUs being able to process multiple computations in parallel, through their usual high amount of cores. Apart from utilizing this characteristic at the edge, which comes with a high cost, other specialized hardware can suffice for real time inference. Companies, e.g., Nvidia, Google, and Asus, offer a wide array of products to enable highly efficient AI-based applications at the edge of the network. Nvidia offers the Jetson Portfolio[5] consisting of computing modules and single board computers. These include the Jetson Nano, TX2, Xavier NX, and more, each offering different levels of computing power. In addition, they offer an extensive development kit, called JetPack SDK, which is advertised as the future for development of AI-based applications for embedded systems, shipping with CUDA, cuDNN and TensorRT support. These libraries allow for model optimization and improved execution time of neural networks on Nvidia GPUs.

Google offers the Coral product line[6], built around their Edge TPU - a processor built and designed solely for the execution of neural networks. A single board computer and USB-dongle advertised for prototyping utilize this Edge TPU, to accelerate machine learning models based on TensorFlow Lite. As previously mentioned in Section 2.3.2, TensorFlow Lite comes with limited operator compatibility, which requires careful selection of operations during development. This is aggravated by the Edge TPU, as even less operations are supported. One noteworthy operator is *LeakyReLU*, an activation function which was presented in Section 2.2.3, which is widely used in machine learning models, but not yet supported by the Edge TPU. Before performing inference, TensorFlow models have to be converted to 8-bit quantized TensorFlow Lite models, which will be explained

[2] https://www.tensorflow.org/
[3] https://pytorch.org/
[4] https://www.tensorflow.org/lite/guide
[5] https://www.nvidia.com/en-us/autonomous-machines/
[6] https://coral.ai/products/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

in Section 2.3.4, and converted with the Edge TPU Compiler provided by Google. The workflow of such a process is visualized in Figure 2.7

Figure 2.7: Conversion pipeline to generate a model for inference on an Edge TPU

Asus also offers products around machine learning with the Tinker Board product line, which mainly focuses on the deployment of Machine Learning models at the edge. Most notable are two Single Board Computers, the *Tinker Edge R*, featuring the Rockchip RK3399PRO Processor, and the *Tinker Edge T* featuring the Edge TPU of the previously mentioned Google Coral product line.

2.3.4 Quantization

Quantization is an optimization mechanism to speed up the performance of machine learning models, while also reducing their size. This can facilitate real-time performance of machine learning models on resource constrained devices, as performance boosts of up to 3x and a size decrease of up to 4x can be achieved with this procedure for TensorFlow Lite models.[7] Quantization can either be done during or after training, with the former being called *Quantization-aware training* and the latter *Post-training quantization*.

Model parameters, such as weights, biases or activations, which are learned during training, as described in Section 2.2.4, are typically stored in 32-bit floating point numbers. Both variants of Quantization convert these values to a selection of different types, such as 16-bit floating point numbers, 8-bit integer numbers or a dynamic range of numbers.

One drawback of this is that the quantized model may achieve lower accuracy in comparison to the original model. Predicting the impact quantization may have on accuracy is difficult, as it depends on various factors, such as model type, calibration data, and more.

Full Integer Quantization

To achieve full integer quantization, all model parameters, such as biases, weights, activations and model output, have to be converted to 8-bit integer numbers. For this, the range of all floating point numbers has to be calibrated with a representative dataset of input data, by performing inference on a subset of training or validation data.

[7] https://www.tensorflow.org/lite/performance/post_training_quantization

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

2.3.5 Collaborative Inference

Offloading computation to edge devices is widely popular within the edge computing paradigm, because it solves many problems of offloading computation to cloud servers, e.g., latency, communication, and energy consumption. Offloading itself can be further divided into full offloading and partial offloading, as defined by Mach et al. [MB17], in which computation is either fully or partly offloaded to a device on the edge of the network. These concepts can also be applied to inference, as multiple papers [TMK17] [KHG+17] [LZC18] deal with the partial and full offload of inference tasks to improve aspects such as performance, resource utilization, and communication, when dealing with AI-based applications. This heavily ties into *AI on edge*, as defined in Section 2.3.1, mainly dealing with the framework design itself. The basic idea is to utilize the layer-based characteristics of Convolutional Neural Networks (CNNs), which were further elaborated in Section 2.2.5, to split up machine learning models into two separate models. Inference on the head (first half of the) model can then be executed in proximity to, or on, the edge device generating data, while the tail (second half of the) model is deployed on a server, waiting to receive data and finish inference. Instead of raw data, intermediate features are transmitted from the edge device to the server. This partial offloading approach is pictured in Figure 2.8 next to the more traditional no-offloading and full-offloading approaches. Other names for partial offloading include Deep Neural Network partitioning, distributed Deep Neural Network, model splitting, or Collaborative Inference.

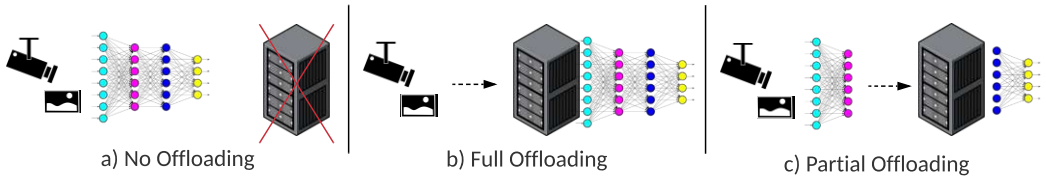

Figure 2.8: Offloading Categories, adapted from [KHG+17]

Partition Point

As communication cost is an important topic for edge computing in general, partial and full offloading solutions have to consider their bandwidth usage during system design. While bandwidth usage for full offloading solutions are trivially calculated, Co-Inference systems can be adjusted to achieve the lowest possible communication cost, as the partition point of the model determines how much data will have to be transmitted.

Selecting the ideal partition point is difficult because of various peculiarities of deep neural networks. Through the nature of CNNs, transmission of intermediate features can entail increased communication cost when compared to the original input. This is due to convolutional layers commonly applying multiple kernels, which was further elaborated in Section 2.2.5, resulting in a larger output size when compared to the original input.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Additionally, CNNs can not be split anywhere, as e.g., deeper layers might require the output of a previous layer, without being the direct descendant. Commonly known as *route layers*. Such a neural network is depicted in Figure 2.9, which is partitioned at the dotted line in red. The server receiving intermediate features of *layer 3* would further require the output of *layer 2* to finish inference.

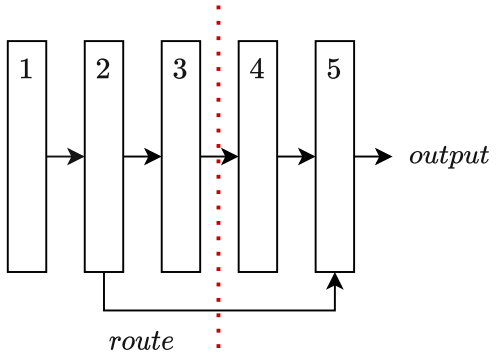

Figure 2.9: CNN with route layer and partition point in red

In addition to transmission cost, computational overhead associated with each additional layer may also influence the decision of a partition point in Co-Inference. When comparing runtime of single layers, possible bottleneck layers can be offloaded to more powerful servers in partial offloading scenarios, by partitioning the model before this bottleneck layer. Figure 2.10 shows an analysis of layer runtime and transmission cost (*output data size*) of AlexNet - a popular CNN. With this analysis, layer *conv_2* can be identified as a possible bottleneck layer, due to its high runtime. Therefore, selection of an adequate partition point for a Co-Inference system may require careful analysis of both transmission cost and computational overhead of layers within a neural network.

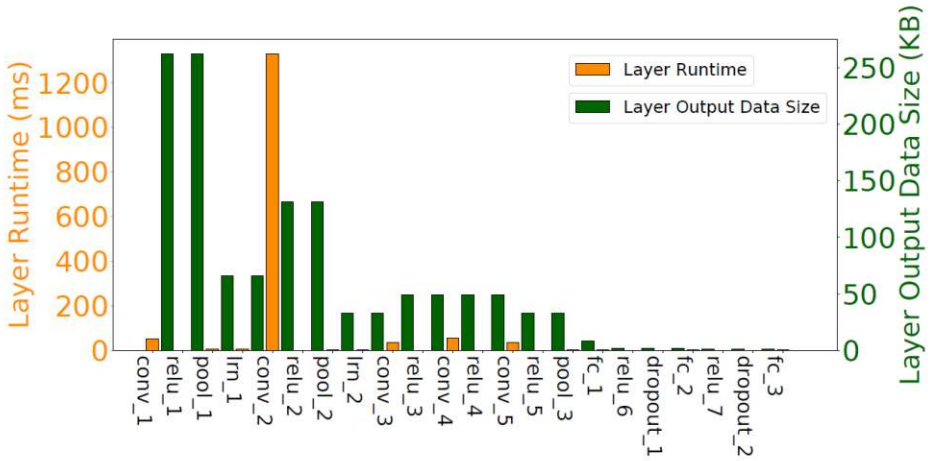

Figure 2.10: AlexNet layer inference and transmission cost analysis, by Li et al. [LZC18]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Privacy

As the primary goal of Co-Inference is to increase performance of existing machine learning models, privacy of intermediate features and transmission thereof is often overlooked. Privacy has become a great concern when utilizing AI in recent years, which can be mainly attributed to the GDPR [Cou22] and the prevalence of AI-based applications in many areas of the world. The privacy concern for Co-Inference systems is simple: What can an adversary achieve by gaining access to intermediate features? In comparison to typical offloading scenarios, where raw video material could be unlawfully processed by the receiving server, intermediate features present a new challenge with privacy concerns.

He et al. propose different *"attacks to compromise the inference data privacy in collaborative deep learning systems"* [HZL19] and present multiple scenarios on how an adversary may be able to turn intermediate features into original images. They ask a similar question to above, which is *"if one intermediate participant is compromised and controlled by an adversary, can he recover an arbitrary input sample?"* [HZL19]. They group their attacks into three categories: white-box, black-box and query-free attacks. These attacks differ in their requirements, e.g., white-box attacks require the knowledge of the parameters and structure of the original model, which has produced the intermediate features. The results of these attacks and their impact on privacy differ based on the machine learning model and its partition points, and have to be analyzed individually.

2.4 Data Protection and Personally Identifiable Information

With the emergence of the General Data Protection Regulation (GDPR) [Cou22] in 2016, privacy has moved further in becoming a human right. The GDPR mandates that *"personal data shall be processed in a manner that ensures appropriate security of the personal data, including protection against unauthorised or unlawful processing and against accidental loss, destruction or damage, using appropriate technical or organizational measures ('integrity and confidentiality')."* [Cou22, Ch. II, Art. 5, 1.(f)]. This has a significant impact on how systems have to be developed. As most technology sectors deal with personal data, additional time and resources have to be invested into correctly implementing state-of-the-art security and privacy solutions. This also necessitates the definition of what counts as personally identifiable information (PII). Numerous definitions of PII can be found in literature, while this thesis will use the definition of personal data, given by the EU in the GDPR [Cou22, Ch. I, Art. 4, (1)]:

"'personal data' means any information relating to an identified or identifiable natural person ('data subject'); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person;"

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

2.4.1 Video Analysis Pipelines

Popular Edge Intelligence applications include Video Analysis Pipelines. There exist various use cases in smart cities, e.g, identifying problematic areas related to traffic in the city, self-driving cars being assisted through smart traffic lights, or surveillance. These scenarios require very low latency, ranging from a second to tens of milliseconds, requiring a geo-distributed edge infrastructure as described by Ananthanarayanan et al. [ABB+17]. They also point out the resource-accuracy relationship, in which higher accuracy needs more resources and vice-versa. This is especially challenging in highly distributed settings, as multiple powerful edge nodes are necessary to fulfill latency requirements. With the requirements of modern machine learning models decreasing over the past years, utilizing them at the edge of the network becomes a real possibility as the tools and hardware, which were outlined in Section 2.3.1, promise to bring inference to the very edge.

Privacy in Video Analysis Pipelines

With the widespread use of closed-circuit television cameras worldwide, Senior et al. [SPH+05] predicted that the analysis of these video streams could be widely automated through algorithms, instead of requiring manual labor, in 2005. Considering the current progress in computer vision, this scenario is now reality, as AI can analyze video streams in real time, generating valuable insights. This necessitates that security and privacy concerns related to video analysis pipelines, which include surveillance systems, are dealt with in accordance to the GDPR.

Senior et al. [SPH+05] further propose a privacy protecting video console, also outlining six aspects important for privacy in video surveillance systems:

- What data is present?
- Has the subject given consent?
- What form does the data take?
- Who sees the data?
- How long is data kept?
- How raw is the data?

Privacy concerns which are exacerbated with the introduction of the GDPR, require that these aspects, among others, are dealt with in Video Analysis Pipelines.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 3

Impact on Performance and Privacy of Partition Points in a Prototypical Co-Inference VAP

This chapter explains the software components that were implemented in order to answer the research questions of this thesis. First, an overview is given, outlining the needed components to implement a prototypical Co-Inference VAP. Following sections explain each of the required components in detail. Lastly, the implemented image reconstruction attack necessary to evaluate the impact of partition points on privacy is presented.

3.1 Overview

To evaluate the performance and privacy of different partition points in a Co-Inference VAP the following components are needed: A Co-Inference VAP has to be implemented, to be able to measure the performance with different benchmarks, such as throughput, resource utilization, and required bandwidth. To compare the performance of the Co-Inference VAP to a baseline, a locally executed VAP has to be implemented and evaluated as well. For the Co-Inference VAP, multiple partition points have to be determined, to allow for the evaluation of their impact on performance and privacy. First, an adequate machine learning model, i.e., a model which can be split, has to be selected, as not every model can be split at will. Second, to enable inference on the testbed, additional constraints, such as operator compatibility, which was further elaborated in Section 2.3.3, have to be considered for the model. Third, a quantization step for the model is required to support test device accelerators and improve performance.
After implementation of both VAPs, the evaluation of impact on privacy for each partition point can be carried out. The different partition points evaluated for performance will be used in an image reconstruction attack. This image reconstruction attack aims

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

to reconstruct the original image from intermediate features. Therefore, intermediate features have to be produced with each partition point. These intermediate features can then be used in the image reconstruction attack, resulting in the generation of an image. As multiple partition points are evaluated, images will be generated for each of the partition points. The quality of these generated images is then quantified with a specific metric, to measure how similar the generated images are to the original image. Based on this metric, the effects of partition point on privacy in Co-Inference can be evaluated.

3.2 Video Analysis Pipeline Setup

A VAP performing the task of object detection locally was implemented, which is referred to as Local Inference VAP in the remainder of this thesis. The machine learning model chosen for this task was YoloV3-Tiny, as YoloV3 is one of the best performing and most accurate models for object detection [RF18]. The tiny version of YoloV3 was chosen for the VAP, as its small size allows efficient execution on resource constrained devices, which are typical for edge computing infrastructures. For example, an Nvidia Pascal Titan X can process 35 frames per second with YoloV3, while it achieves up to 220 frames per second for YoloV3-Tiny [RF22, RF18].
The VAP used a GitHub repository[1] from *qqwweee* as its codebase, as it provides a Keras implementation of YoloV3-Tiny. Python 3.7.5 and TensorFlow 2.3.1 were used to implement the VAP. Because the test devices included the *Tinker Edge T*, featuring an Edge TPU, which was presented in Section 2.3.3, the model had to be adjusted for testing purposes. The *LeakyReLU* activation function is used in standard configurations of YoloV3-Tiny, but not supported by the Edge TPU. Therefore, the activation function had to be replaced with ReLU. Another GitHub repository[2] by *guichristmann*, builds upon the aforementioned repository, and offers YoloV3-Tiny with ReLU as the activation function. It was integrated into the VAP instead and will be referred to as *YoloV3-Tiny* in future sections as well, as only the activation function has been changed.

To ensure equal conditions for the performance evaluation of the test devices, which will be presented in a later section, a single video was chosen as the input file for testing purposes. Table 3.1 displays the properties of the sample video. The video shows people walking over a street. Hence, the object detection algorithm of the VAP should detect objects of the type *person* in the video. A sample frame of the analysis on the sample video can be seen in Figure 3.1.

Width	Height	Channels	Duration	Frames
596 px	336 px	3	10 s	273

Table 3.1: Properties of the sample video used for evaluation of the VAP

[1] https://github.com/qqwweee/keras-yolo3
[2] https://github.com/guichristmann/keras-yolo3

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Figure 3.1: A single frame of the implemented VAP with YoloV3-Tiny - sample video from Shutterstock [JL22]

To understand how the Local Inference VAP works, an overview of the steps applied to every frame during local execution is depicted in Figure 3.2. The first step is *resizing* the input to be compatible with the machine learning model, if necessary. YoloV3-Tiny has a fixed input size of 416×416×3 pixels, which required resizing of the input video before inference could be performed. Then, the resized input is fed to the model, which performs *inference* and returns predictions based on what was learned during training. For YoloV3-Tiny the resulting predictions took the following shape:

```
[[[263.0816,  34.8289], [413.7800, 340.6221]], ...],
[0.989391148090, ...],
[0, ...]
```

The first array held coordinates of bounding boxes, the second array the confidence scores of the corresponding bounding box, and the third array the values of the predicted classes. For the above example, *0* would correspond to the *person* class, with the model being $\sim 98.9\%$ confident that there is a person within the calculated bounding box. The last step is to *post-process* the predictions from inference, which can vary depending on the use case. In the implemented VAP the predictions were turned into rectangles, including the related class and confidence score, which were drawn over the original frames to produce the output visible in Figure 3.1. The above steps had to be repeated for each frame in the video, to fully process the video with the VAP.

These steps are commonly used in VAPs and are not exclusive to the implemented VAP of this thesis.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

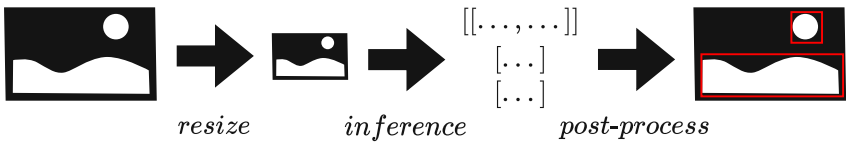

Figure 3.2: Process of the Local Inference VAP for each frame

3.3 Model Partition

To implement a Co-Inference VAP, the chosen model YoloV3-Tiny had to be split into two models. These parts, commonly referred to as *head* and *tail* model, were then deployed on two separate devices - and edge device and server. Edge devices represent typical resource-constrained devices, while edge servers are more powerful and therefore more expensive devices. After resizing the input image, intermediate features were generated with the head model on the edge device and subsequently sent to the edge server. The edge server then finished inference with the tail model and returned the inference results to the edge device. At last, the edge device applied the same post-processing step as the Local Inference VAP, i.e., adding rectangles, class, and confidence score to the original image. The full process of the Co-Inference VAP is depicted in Figure 3.3.

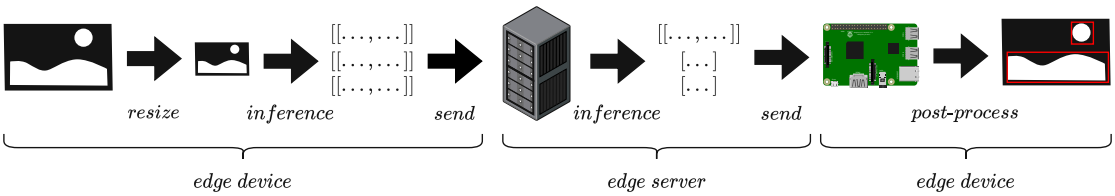

Figure 3.3: Process of the Co-Inference VAP for each frame

As previously stated in Section 2.3.5 choosing a partition point is not trivial, which is why YoloV3-Tiny and its candidate partition points will be presented in this section. The network architecture of YoloV3-Tiny can be found in Figure 3.4. Additionally, a tabular specification of the architecture of YoloV3-Tiny can be found in Figure 3.5. Both of these specifications also apply to YoloV3-Tiny with ReLU as the activation function, which was further elaborated in Section 3.2. YoloV3-Tiny consists of five types of layers:

1. convolutional layers
2. max pooling layers
3. two route layers
4. one up-sampling layer
5. two YOLO detection layers

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

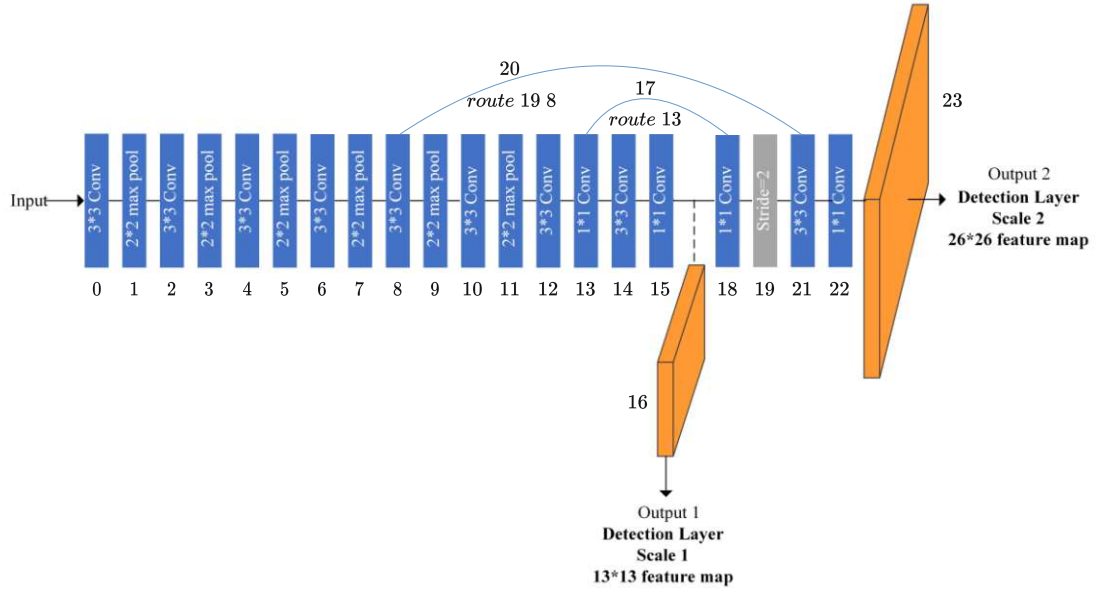

Figure 3.4: YoloV3-Tiny architecture, adjusted from [FWR19]

As previously mentioned in Section 2.3.5, route layers hinder easy partition of machine learning models. This holds true for YoloV3-Tiny as well, as two route layers are present within the network. To successfully split this model after *route 19 8* (see Figure 3.4), e.g., after *layer 11*, finishing inference on the receiving edge server would require the output of *layer 8* in addition to the output of *layer 11*. Similarly, splitting the model after *route 13*, e.g., after *layer 14*, would require the intermediate features of *layers 8, 13, and 14* to be included in the transmitted data. This would increase bandwidth usage and add additional complexity to the Co-Inference VAP. Furthermore, since the image reconstruction attack will be performed on intermediate features, including those of *layer 8* in the transmitted data, would essentially allow reconstructing from *layer 8*. Because of these two reasons, partition points requiring the inclusion of intermediate features from *layer 8* in the transmitted data are not included in the analysis. Only two layers after *route 19 8*, *layer 21* and *layer 22*, allow to circumvent the transmission of intermediate features from *layer 8*. As these partition points are located at the end of the network, no real improved performance can be expected, therefore they were not included in the analysis. With this reasoning, every layer up to, and including, *layer 8*, was selected as a partition point for the analysis. This results in 9 possible partition points, partitioning the model *after* every layer numbered from 0 to 8.
These partition points are referred to as *conv_n* and *pool_n*, denoting the type of last layer included in the partition point - either a *convolutional* or *max pooling* layer. The number n included in the label directly corresponds to the n-th convolutional or max pooling layer of Figures 3.4 and 3.5. Note that the batch normalization and ReLU layers were also included in the partition points ending on a convolutional layer. Resulting partition points and their corresponding layer from Figure 3.4 are described in Table 3.2.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Partition point	Corresponding Layer in Figure 3.4
conv_0	layer 0
pool_0	layer 1
conv_1	layer 2
pool_1	layer 3
conv_2	layer 4
pool_2	layer 5
conv_3	layer 6
pool_3	layer 7
conv_4	layer 8

Table 3.2: Selected partition points for YoloV3-Tiny

Layer	Type	Filters	Size/Stride	Input	Output
0	Convolutional	16	3 × 3/1	416 × 416 × 3	416 × 416 × 16
1	Maxpool		2 × 2/2	416 × 416 × 16	208 × 208 × 16
2	Convolutional	32	3 × 3/1	208 × 208 × 16	208 × 208 × 32
3	Maxpool		2 × 2/2	208 × 208 × 32	104 × 104 × 32
4	Convolutional	64	3 × 3/1	104 × 104 × 32	104 × 104 × 64
5	Maxpool		2 × 2/2	104 × 104 × 64	52 × 52 × 64
6	Convolutional	128	3 × 3/1	52 × 52 × 64	52 × 52 × 128
7	Maxpool		2 × 2/2	52 × 52 × 128	26 × 26 × 128
8	Convolutional	256	3 × 3/1	26 × 26 × 128	26 × 26 × 256
9	Maxpool		2 × 2/2	26 × 26 × 256	13 × 13 × 256
10	Convolutional	512	3 × 3/1	13 × 13 × 256	13 × 13 × 512
11	Maxpool		2 × 2/1	13 × 13 × 512	13 × 13 × 512
12	Convolutional	1024	3 × 3/1	13 × 13 × 512	13 × 13 × 1024
13	Convolutional	256	1 × 1/1	13 × 13 × 1024	13 × 13 × 256
14	Convolutional	512	3 × 3/1	13 × 13 × 256	13 × 13 × 512
15	Convolutional	255	1 × 1/1	13 × 13 × 512	13 × 13 × 255
16	YOLO				
17	**Route 13**				
18	Convolutional	128	1 × 1/1	13 × 13 × 256	13 × 13 × 128
19	Up-sampling		2 × 2/1	13 × 13 × 128	26 × 26 × 128
20	**Route 19 8**				
21	Convolutional	256	3 × 3/1	13 × 13 × 384	13 × 13 × 256
22	Convolutional	255	1 × 1/1	13 × 13 × 256	13 × 13 × 256
23	YOLO				

Figure 3.5: YoloV3-Tiny network structure table, from [HHW+19]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

3.4 Quantization

Quantization was primarily carried out to improve inference speeds. Additionally, the Edge TPU accelerator of one of the test device, required a full integer quantized TensorFlow Lite model. Therefore, the YoloV3-Tiny model, and all head and tail models of the partition points of Table 3.2, were 8-bit integer quantized. As the testbed also included Nvidia devices, quantization was carried out with the *TensorFlow-TensorRT (TF-TRT) integration* for these devices. For the TensorFlow Lite quantization, the *TensorFlow Lite* converter was used. Compiling the 8-bit quantized TensorFlow Lite model for the Edge TPU was executed with the *Edge TPU Compiler*. To provide the representative dataset, which was further elaborated in Section 2.3.4, 500 random images of the Pascal VOC 2007 [The22] test data were used. For the quantization of the head models and YoloV3-Tiny model, the approach was straight forward, as only the images had to be supplied to the converters for calibration. Tail models however required intermediate features as calibration input instead of images. These intermediate features were generated with the related head model of each tail model. The resulting models are presented in Table 3.3. YoloV3-Tiny and its quantized version were used in the Local Inference VAP. The head and tail models were for the Co-Inference VAP. Types of the resulting models include TensorFlow, TensorFlow Lite, TensorFlow Lite 8-bit quantized, TF-TRT 8-bit quantized, and Edge TPU 8-bit quantized models. These model types support different kind of accelerators, listed in the column *Accelerator*, which are important for compatibility with the devices found in the testbed.

Type	**YoloV3-Tiny**	**Head models**	**Tail models**	**Accelerator**
TensorFlow	✓	✓	✓	No accelerator, non-ARM CPU
TensorFlow Lite	✓	✓	✓	No accelerator, ARM CPU
TensorFlow Lite 8-bit	✓	✓	✓	No accelerator, ARM CPU
TF-TRT 8-bit	✓	✓	✓	Nvidia GPU
Edge TPU 8-bit	✓	✓	✓	Edge TPU

Table 3.3: Available models for evaluation

3.5 Image Reconstruction Attack

This section presents the approach that was taken to perform and evaluate the image reconstruction attack. First, a theoretical setting will be given, which describes how the attack could be executed in the real world. Then, the attack and objective evaluation of its success will be explained.

3.5.1 Theoretical Setting

The Co-Inference VAP of Section 3.2 is implemented and deployed on various edge devices and servers in a city. This VAP aims to generate alarms in case of a traffic accident, to automatically alarm emergency services and operators of the system. Therefore,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

edge devices are distributed on various intersections and equipped with cameras, while more powerful edge servers are deployed in a datacenter. One of the operators wants to unlawfully process the transmitted data. In the case of traditional offloading, where the raw data is visible to the receiving edge server, this task could be easily achieved by the operator. Even with an encrypted transmission, the operator would be able to decrypt transmitted data, as the server has to be equipped with the corresponding key to be able to perform inference. To mitigate these privacy concerns, the VAP implements Co-Inference, which transmits intermediate features instead of raw video data. Obtaining these intermediate features is of no immediate gain, as they can not be easily turned into the original video. The adversary performs the image reconstruction attack presented by Oh et al. in [OL19]. Depending on the results of this attack, i.e., if personally identifiable information can be extracted from the reconstructed video, the adversary will either be satisfied or disappointed.

The results of the reconstruction attack depend on the partition point of the Co-Inference VAP, which is one of the major factors thriving this thesis. By executing the image reconstruction attack on different partition points of YoloV3-Tiny, the impact of partition point on reconstructability and therefore privacy can be shown. If possible, a partition point will be found, which is not prone to the image reconstruction attack, thus protecting the privacy of individuals visible on the original video data.

3.5.2 The Attack

To reconstruct images from the intermediate features of the different partition points from Table 3.2, a white-box attack was used, as outlined by Oh et al. in [OL19]. Executing this attack in a real world scenario requires the adversary to discover which machine learning model is used within the Co-Inference VAP. As the VAP used YoloV3-Tiny, identifying the model would be manageable, by analyzing the properties of obtained intermediate features, e.g., dimensions of the transmitted data. Figure 3.5 shows dimensions of the output produced by each layer of the used model, which could be used to identify the specific layer of the YoloV3-Tiny network used as partition point in the Co-Inference VAP. Then, the adversary has to obtain the machine learning model used within the VAP. Obtaining this model would be manageable as well, as multiple online sources exist where it can be downloaded. After determining what layer is being used for transmission, the obtained machine learning model has to be split at the same partition point.
The necessary knowledge required for this approach is summarized, with added notation based on the outline of [OL19]:

- f^l - Head portion of original model up to layer l, which generated the obtained intermediate features.
- $f^l(\hat{x})$ - Obtained intermediate features, with $\hat{x}$ denoting the original image that has to be reconstructed.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Following is an overview of how the approach works, from [OL19].

1. Generate a randomized input variable x as input for the model partition f^l, requiring the correct dimensions (416×416×3 for YoloV3-Tiny)

2. Pass the input variable x to f^l, generating intermediate features $f^l(x)$

3. Compute loss L_R, which is the $L2$ loss between the generated intermediate features $f^l(x)$ and obtained intermediate features $f^l(\hat{x})$: $L_R = \sum_{l \in L}(f^l(x) - f^l(\hat{x}))^2$. This process is also described visually in Figure 3.6.

4. Compute the total variation loss $L_{TV} = \sum_{i,j}((x_{i,j} - x_{i+1,j})^2 + (x_{i,j} - x_{i,j+1})^2)$, with $x_{i,j}$ denoting the (i,j)-th pixel value of x.

5. Optimize the image variable x^* by minimizing the objective function $x^* = argmin_x(L_R + \lambda L_{TV})$ through backpropagation.

6. Repeat from step 2, with the adjusted image variable from step 5, for y iterations.

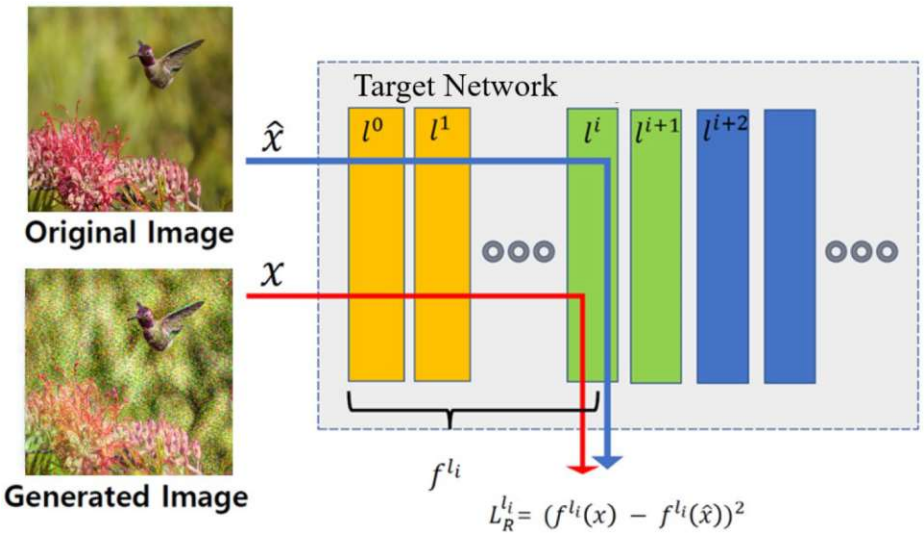

Figure 3.6: Calculation of image reconstruction loss of i-th layer, from [OL19].

Similar to Oh et al. [OL19] the learning rate has been set to $5 * 10^1$, lambda of the objective function to $1 * 10^{-3}$, number of iterations y to 500 and Adam was used as the optimizer. The image reconstruction attack was implemented with Python 3.7.5 and TensorFlow 2.3.1. Ten images were used in the image reconstruction attack for each partition point, resulting in a 10x10 matrix of original images and their reconstructed versions of each partition point.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

3.5.3 Similarity Metrics

To quantify the similarity between the generated and original image, multiple state-of-the-art metrics were used. Peak signal-to-noise ratio (PSNR) is a commonly used metric to *"measure the pixel level recovery quality of the image"* [HZL19] on compressed image or video data. Structural Similarity Index (SSIM) is also used to measure the similarity between two images, and *"is considered to be correlated with the quality perception of the human visual system"* [HZ10]. Additionally, image difference (D) has been used, which has been defined by Oh et al. [OL19] as per Formula 3.1.

$$D(x, \hat{x}) = log(\frac{1}{HWC} \sum_{i,j,c} |x_{i,j,c} - \hat{x}_{i,j,c}|^2 + 1) \tag{3.1}$$

H, W, C denote the height, width and channels of the image, with *i,j,c* denoting the (i, j, c)-th pixel of both generated image x and original image $\hat{x}$. When $D(x, \hat{x})$ is zero, the images are identical. Additionally, D_{batch} is defined as *"the mean value of image difference among a set of images"* [OL19].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 4

Methodology

This chapter outlines the methodological approach that was carried out within this thesis to answer the research questions. First, the two underlying research questions of the thesis will be revisited. Second, the test environment needed to evaluate the performance of different partition points in Co-Inference will be described. Third, an explanation on how the performance of the VAP was benchmarked is given. Lastly, the methodology used to measure the impact of different partition points on privacy will be discussed.

4.1 Research Questions

- **RQ$_1$**: How do the different possible partition points of a Co-Inference VAP impact the performance in regard to throughput, resource utilization, and required bandwidth?
- **RQ$_2$**: How do the different possible partition points of a Co-Inference VAP impact security and data protection?

4.2 Evaluation Environment

To allow for an evaluation of a Co-Inference VAP in a real world scenario, an infrastructural environment typical for edge computing was established. It comprises a wide array of devices, allowing for multiple possible system compositions, reflecting the heterogeneous nature of edge computing. These devices were grouped into two categories based on their computing power, namely edge devices and edge servers. While edge servers are characterized by their high computing power and associated with high cost, edge devices represent typical resource constrained devices. With this environment the performance of different edge devices and servers can be compared. Additionally, the performance of the different partition points for each of the edge devices and servers can be compared in

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

order to answer RQ_1. This testbed is presented in Section 4.2.1. As the intermediate features produced by the different partition points are essentially the same for all test devices, the testbed is not needed for evaluation of the image reconstruction attack. Instead, the image reconstruction attack was performed on a single test device, i.e., the Desktop PC.

4.2.1 Testbed Overview

This section presents the devices used for the evaluation of the Co-Inference VAP and its possible partition points. Additionally, to allow a comparison of Co-Inference benchmarks to a baseline, the performance of the Local Inference VAP was benchmarked for all these devices as well. These test devices and their specifications can be found in Table 4.1. Next to the device name, properties important for the execution of the machine learning model, such as CPU, accelerator and available RAM are included. The last column features a short identifier, which is used in later sections to reference specific devices. Furthermore, the two categories, edge devices and servers, are clearly separated in the table.

Device	CPU	RAM	Accelerator / GPU	Identifier
Edge Devices				
Raspberry Pi 3B+	4x Cortex-A53 @ 1.4 GHz	1 GB	-	RBP
Nvidia Jetson TX2	4x Cortex-A57 @ 1.9 GHz	8 GB	256-core NVIDIA PASCAL	TX2
Nvidia Xavier NX	6x NVIDIA Carmel @ 1.9 GHz	8 GB	384-core NVIDIA Volta	XNX
Tinker Edge T	4x ARM A53 @ 1.5 GHz	1 GB	Google Edge TPU	TET
Edge Servers				
ASRock 4x4Box-4800U	AMD Ryzen 7 4800U @1.8 GHz	16 GB	-	ASR
Desktop PC	Intel i5-8400	16 GB	Nvidia GTX 1060 6 GB	DPC

Table 4.1: Testbed devices

All devices featuring an Nvidia accelerator, i.e., Desktop PC, Xavier NX, and Jetson TX2 had CUDA 10.2, TensorRT 8.0.1, and cuDNN 8.2.1 installed.

4.3 Benchmarking Performance

This section explains how the performance of the Co-Inference VAP and possible partition points were benchmarked. As previously depicted in Figure 3.3, the steps to process a single frame in the Co-Inference VAP can be described as follows. After performing resizing and inference on the edge device, intermediate features are transferred to the edge server, which performs inference and returns the results. Afterwards, the edge device can post-process these, by e.g., drawing bounding boxes onto the original image. The milliseconds needed for each of the above steps have to be recorded to benchmark the performance. Table 4.2 shows at which step each of the collected timestamps were recorded. Table 4.3 shows how the timings of each step can be calculated, based on Table 4.2.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Timestamp	Steps in Co-Inference VAP
t_0	Before resizing
t_1	After resizing
t_2	After edge device finishes (head model) inference
t_3	After edge server receives intermediate features
t_4	After edge server finishes (tail model) inference
t_5	After edge device receives full inference results from the edge server
t_6	After performing post-processing

Table 4.2: Timestamps and respective steps of the Co-Inference VAP

Step	Calculation
resize	t_1-t_0
head model inference	t_2-t_1
tail model inference	t_4-t_3
latency	t_3-t_2 + t_5-t_4
post-process	t_6-t_5

Table 4.3: Co-Inference VAP benchmark calculation

The performance boost Co-Inference and different partition points can yield is one of the two major aspects investigated in this thesis. Latency, see Table 4.3, depends on multiple factors, such as signal strength, signal quality, interference, or simply heterogeneous hardware infrastructures like cable type, networking devices, and many more. Therefore, it is out of scope of this thesis. Similarly, the post-processing step is not included in the evaluation as well, as its execution time varies based on use case. For example, the execution time of drawing bounding boxes on an original image will differ from performing additional computation with inference results. The *resize* step will be presented in a separate table for each of the edge devices, as this step is independent of the chosen partition point. Therefore, future graphs only include the *head model* and *tail model* steps, to show differences in inference time needed for each partition point.

The performance of the Co-Inference VAP has been measured for each of the edge devices paired with each of the edge servers. For each test device of Table 4.1 a quantized model from Table 3.3 was chosen considering accelerator compatibility. This device-model pairing is also shown in Table 4.4. Because each of the edge devices was paired up with each of the edge servers and all partition points were evaluated, a total of $2(Servers) * 4(Devices) * 9(Partition\ points) = 72$ benchmarks were generated.

The Co-Inference VAP performed object detection on the sample video, which was outlined in Table 3.1. Hence, benchmarks of a single run include metrics for 273 frames. These measurements have been conducted for each of the different partition points on every edge device and server pair. With these benchmarks, the following comparisons can be drawn. The benchmarks of the different partition points can be compared for a single edge device or server. Additionally, the performance of the different edge devices and

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Model type	Device
	Edge Devices
TensorFlow Lite 8-bit quantized	Raspberry Pi 3B+
Edge TPU 8-bit quantized	Tinker Edge T
TF-TRT 8-bit quantized	Nvidia Jetson TX2, Nvidia Xavier NX
	Edge Servers
TF-TRT 8-bit quantized	Desktop PC
TensorFlow	ASRock 4x4Box-4800U

Table 4.4: Test devices and machine learning model pairing

servers can be compared. Furthermore, the impact of a partition point on performance of an edge device or server can be compared to the impact on other devices.

In addition to the above performance benchmarks for the Co-Inference VAP, the Local Inference VAP was evaluated for the sample video from Table 3.1 and each of the test devices as well. This stems from the fact that literature [LZC18, KHG+17, TMK17] suggests that Co-Inference can outperform local inference. Analyzing the performance of the Local Inference VAP allows to compare it to the performance of the Co-Inference VAP. In the Local Inference VAP, *resizing*, *inference*, and *post-processing*, are the necessary steps as shown in Figure 3.2. The resulting timestamps and calculations for the Local Inference VAP are presented in Tables 4.5 and 4.6 respectively. Similar to the Co-Inference VAP, the post-processing step was not included in the evaluation.

Timestamp	Steps in Local Inference VAP
t_0	Before resize
t_1	After resize
t_2	After inference
t_3	After performing post-processing

Table 4.5: Timestamps and respective steps of the Local Inference VAP

Step	Calculation
resize	t_1-t_0
inference	t_2-t_1
post-process	t_3-t_2

Table 4.6: Local Inference VAP benchmark calculation

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

4.3.1 Resource Utilization and Bandwidth Usage

Similar to performance benchmarks, this thesis also analyzed the resource utilization and required bandwidth of each partition point in the Co-Inference VAP. These resource utilization metrics were also collected for the Local Inference VAP. The CPU and memory usage of the VAP were both recorded after each frame. Required bandwidth, i.e., size of intermediate features, were measured before transmission to the edge server. This size can also be analyzed statically, as the output dimensions of each partition point are known, as visible in Figure 3.5. Additionally, the size of inference results, which were returned from the edge servers after finishing inference, was recorded.

This allows for the comparison of resource utilization of different partition points. Furthermore, it allows comparing resource utilization across the different edge devices. As these benchmarks were also collected for the Local Inference VAP, a direct comparison can be drawn for resource utilization. Bandwidth needs for transmission of intermediate features are static, i.e., the size of transmission is always the same for a partition point. However, the bandwidth required to transfer the inference results from edge servers to the edge devices can fluctuate. This allows to compare bandwidth needs of different partition points.

4.3.2 Additional Performance Benchmarks for all Model Types

Previous benchmarks only included four types of models: TensorFlow 8-bit quantized for the Raspberry Pi 3B+, TensorFlow for the ASRock 4x4, and accelerator-quantized (TF-TRT, Edge TPU) models on the remaining devices. Initial results suggested that the impact on inference of the different partition points depended on the used model type. Therefore, the performance was additionally measured for all different model types on the Xavier NX, with the same methodology as presented in Section 4.3. These model types include: TensorFlow (CPU), TensorFlow (GPU), TensorFlow Lite, TensorFlow Lite 8-bit quantized, and TF-TRT 8-bit quantized, which were presented in Table 3.3. This allowed generating additional insights on how different model types impact the performance for each partition point of the Co-Inference VAP.

4.3.3 Discarding Warm-up Data

Because of slow initial inference speeds, which can be attributed to the model warm-up in TensorFlow[1], the first 25 frames and related measurements were discarded in the results. Therefore, graphs presenting various metrics, only include the benchmarks of 248 frames.

[1] https://www.tensorflow.org/tfx/serving/saved_model_warmup

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

4.4 Reconstructability Benchmarks

The image reconstruction attack outlined in Section 3.5.2 was applied to all nine possible partition points, which were identified in Table 3.2, to answer RQ_2. Eight test images from the Pascal VOC 2007 test data were chosen to act as the original image $\hat{x}$ in the image reconstruction attack. Two more were selected, which were used in the paper of Oh et al. [OL19], to allow for comparison of reconstruction results. These images can be found in Figure 4.1. Therefore, nine different intermediate features $f^l(\hat{x})$ were computed by each head model f^l for each of the ten test images. These intermediate features $f^l(\hat{x})$ and head model f^l were then used in the image reconstruction attack. The similarity of the original and generated images were calculated and recorded with three different metrics, namely D, $SSIM$, and $PSNR$, after each iteration of the reconstruction attack. These metrics were also presented in Section 3.5.3. As the generated image is a tensor during reconstruction, the calculation of these metrics was executed with the JPEG encoded version of the generated image and the original image, which was already JPEG encoded. The average values of the three different metrics were then calculated for each of the partition points, over all images. These average values are referred to as D_{batch}, $mSSIM$ and $mPSNR$. As these values were recorded after each iteration of the reconstruction, their progression can be viewed over the full duration of the image reconstruction attack. Additionally, the total time needed for each reconstruction was measured.

Figure 4.1: Images used in the reconstruction attack - Image 1 to 8 are from the Pascal VOC 2007 test data set, image 9 and 10 were also used in the paper of Oh et al. [OL19]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 5

Results

In this chapter, the results of the conducted experiments, which were executed as described in Chapters 3 and 4, are presented. First, the performance benchmarks of the Local Inference VAP are introduced, which serve as a comparison baseline. Second, the performance benchmarks of the Co-Inference VAP are shown in the next section. Third, both the benchmarks of the Local Inference and Co-Inference VAPs will be compared. In the last section, the results of the image reconstruction attack and calculated image similarity metrics are presented.

5.1 Performance Benchmarks

This section presents the performance benchmarks, which were collected according to the methodology, which was presented in Section 4.3. These benchmarks include both the results of the Local Inference and Co-Inference VAP and are presented separately. Finally, a comparison between benchmarks of the Local Inference and Co-Inference VAP will be made, to allow for later discussion.

5.1.1 Local Inference VAP

The Local Inference VAP was evaluated on all test devices, listed in Table 4.1, as previously described in Section 4.3. Table 5.1 shows the mean time needed to resize the input image in ms, rounded to the nearest decimal point. These values are also valid for the Co-Inference VAP, as the resize step is not impacted by Co-Inference. Figure 5.1 shows the mean time in ms, needed to perform inference on a single frame for each of the test devices. Figure 5.2 shows the mean resource utilization in percentage, during local execution of the VAP for each of the test devices. These results showcase the heterogeneity in resource utilization, resize, and inference time of the Local Inference VAP, across the test devices. As expected, the Desktop PC performed best, only needing

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

6 and 2 ms on average to perform inference and resizing for each frame of the video. The Raspberry Pi 3B+ performed worst, needing 48 ms on average to resize the image, and 1519 ms on average to perform inference on a single frame of the video. Edge devices, which were able to use a quantized and accelerator-supported machine learning model, outperformed the ASRock 4x4 in terms of inference, as the latter had to run inference on the CPU. In terms of resource utilization, the ASRock 4x4 had the lowest average RAM usage, while both NVIDIA devices used more than 10 times the RAM than the ASRock 4x4 in terms of percentage. For CPU usage on the other hand, the ASRock 4x4 had the highest average usage of all test devices with 35%. The CPU usage of the other devices averaged between 17% and 25%.

Device	**Mean resize time**
ASRock 4x4	6 ms
Raspberry Pi 3B+	48 ms
Tinker Edge T	17 ms
Jetson TX2	13 ms
Xavier NX	6 ms
Desktop PC	2 ms

Table 5.1: Average time needed for resizing an image in the Local Inference VAP for each device of the testbed

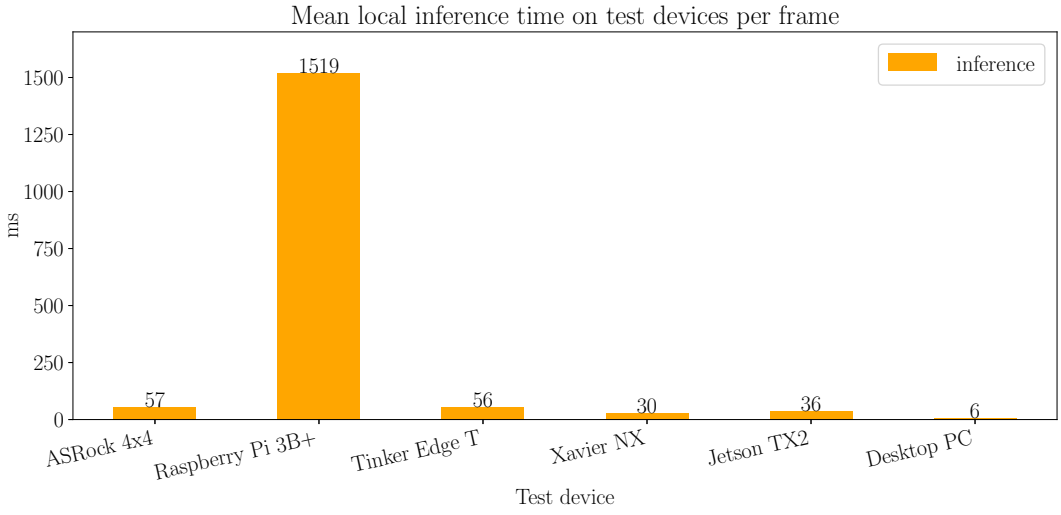

Figure 5.1: Mean local inference time on test devices per frame

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

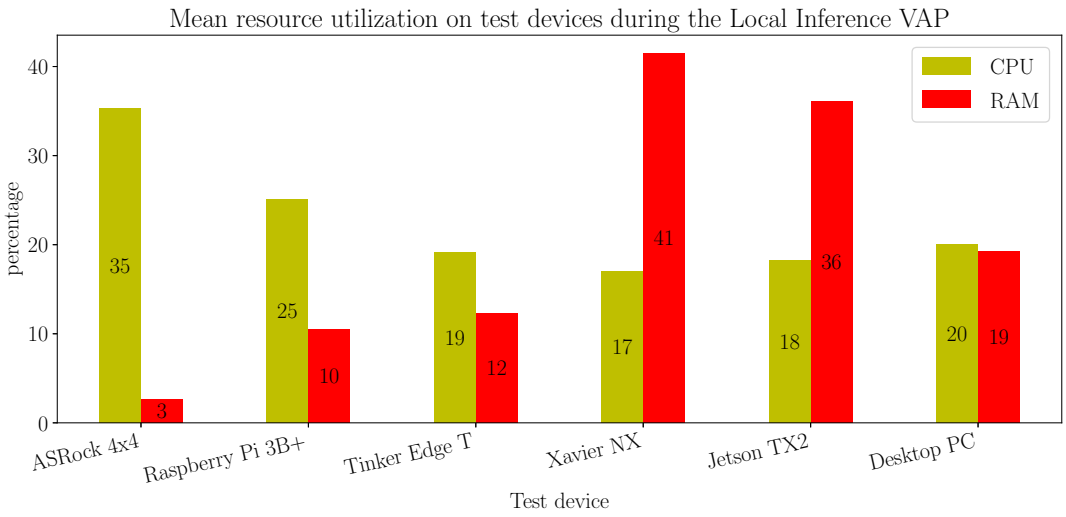

Figure 5.2: Mean resource utilization of the test devices during the Local Inference VAP

5.1.2 Co-Inference VAP

All edge devices were evaluated with all edge servers of Table 4.1 and all partition points of Table 3.2, which resulted in eight different device-server pairs for the Co-Inference VAP. This section is further divided and presents the results based on inference, resource utilization and bandwidth. Different trends have been observed for each of these three metrics in the Co-Inference VAP, which will be summarized in the following sections. Additionally, a list with the device-server pairs that showed the specific trend in the Co-Inference VAP will be given.

Inference

Two trends were identified considering how the different partition points impact the total inference time in a Co-Inference VAP. The total inference time either **increased** or **decreased** with partition point depth. Depth refers to how deep the partition point is in YoloV3-Tiny, which can be seen in Figure 3.4. These trends are demonstrated with two graphs, showcasing the performance benchmarks of two device-server pairs executing the Co-Inference VAP. Figure 5.3 shows the **increasing** trend, which features the Raspberry Pi 3B+ and Desktop PC. Figure 5.4 shows the Tinker Edge T and Desktop PC instead, and demonstrates the **decreasing** trend. A list of the trends for each of the device-server pairs of the Co-Inference VAP evaluation can be found below. These different pairs were abbreviated by using the identifier of the corresponding edge device and server from Table 4.1. This list also includes references to the inference time figures, of which some are presented in the following section, while the remaining can be found in Section A.1.1.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

- **increased**
  - RBP+DPC (Figure 5.3)
  - RBP+ASR (Figure A.1)
- **decreased**
  - TET+DPC (Figure 5.4)
  - TET+ASR (Figure A.2)
  - TX2+DPC (Figure A.3)
  - TX2+ASR (Figure 5.5)
  - XNX+DPC (Figure A.4)
  - XNX+ASR (Figure 5.6)

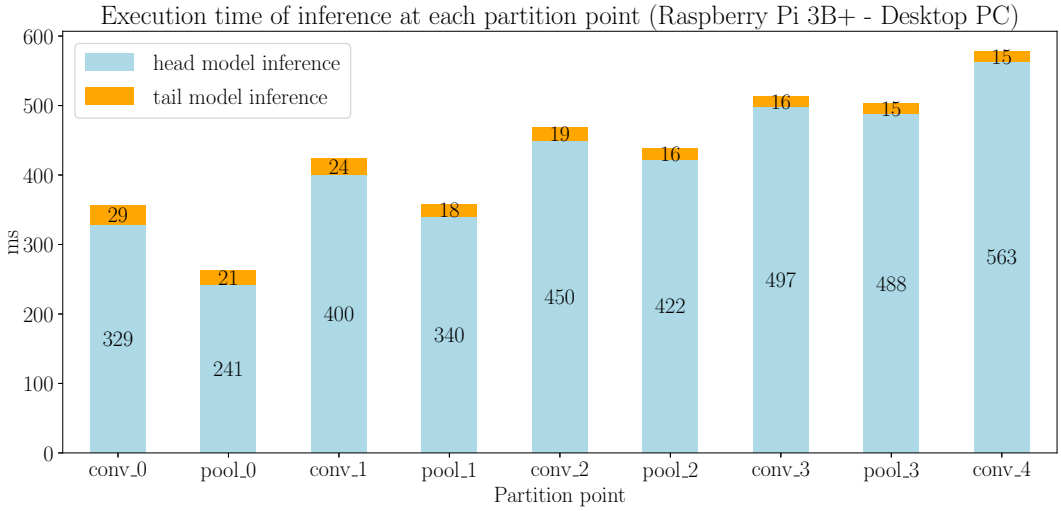

Figure 5.3: Co-Inference execution time on the Raspberry Pi 3B+ and the Desktop PC

Average inference times of the head model on the edge device either **increased** or **decreased** with partition point depth. The above list of trends is also applicable to the head model inference time. Regarding the average tail model inference time on the other hand, only the **decreasing** trend was observed for all edge servers. The Co-Inference VAP was only evaluated with two edge servers, Desktop PC and ASRock 4x4. Figure 5.3 shows the trend of the tail model on the Desktop PC. Figure 5.5 shows the Co-Inference benchmarks of the Jetson TX2 and ASRock 4x4, which showcases the **decreasing** trend for the tail model on the ASRock 4x4.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

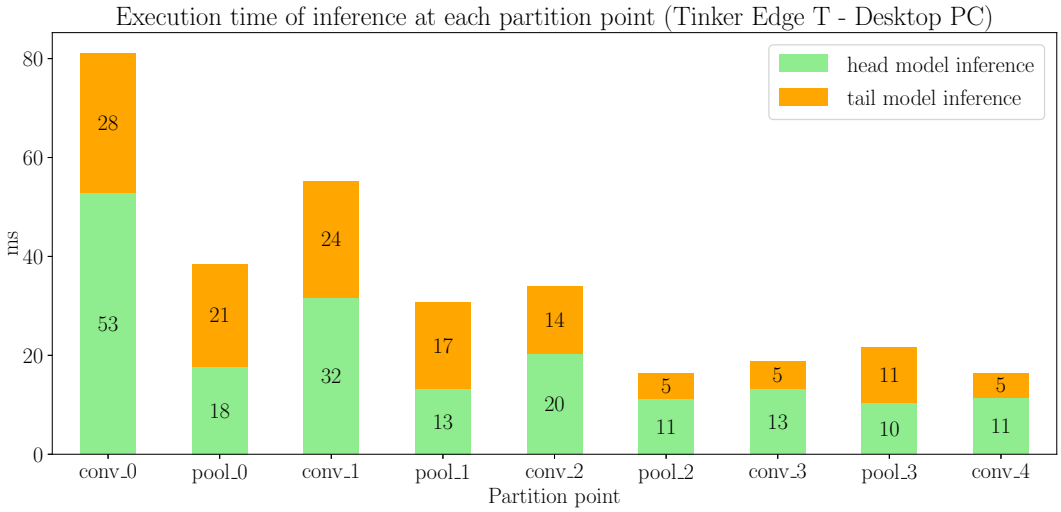

Figure 5.4: Co-Inference execution time on the Tinker Edge T and the Desktop PC

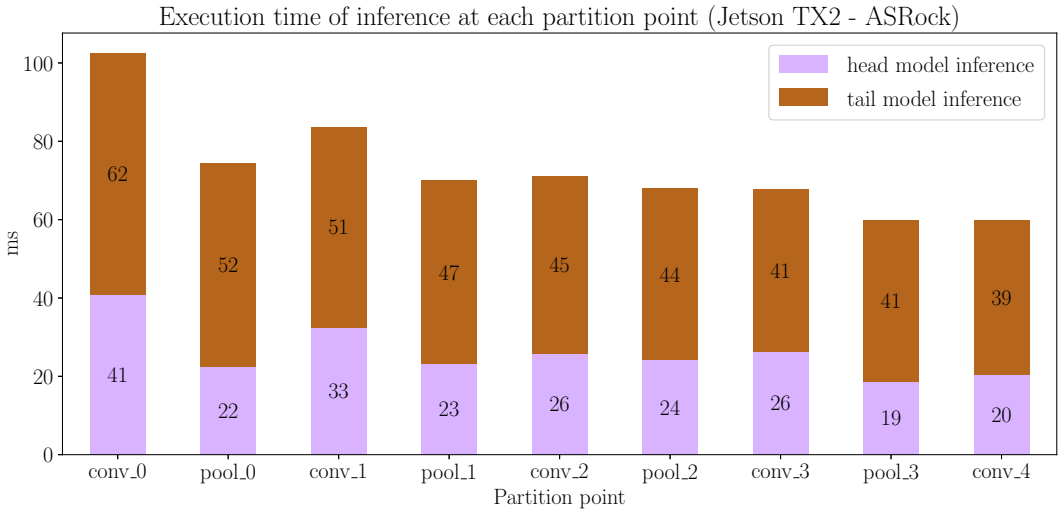

Figure 5.5: Co-Inference execution time on the Jetson TX2 and the ASRock 4x4

In addition to the trends described above, another behavior was observed. When comparing head model inference times of consecutive partition points, the time **alternately** *decreased*, and then *increased* again. This behavior was most obvious in Figure 5.3, which shows the Co-Inference execution time of the Raspberry Pi 3B+ and Desktop PC. Each partition point ending on a *convolutional* layer had a higher inference time, than the following partition point ending on a *max pooling* layer. Following the partition point ending on a *max pooling* layer was another partition point ending on a *convolutional*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

layer with a higher inference time. The following device-server pairs of the Co-Inference VAP evaluation showed this **alternating** behavior in head model execution time for all partition points:

- RBP+DPC, RBP+ASR, TET+DPC, TET+ASR, TX2+DPC

While the following device-server pairs of the Co-Inference VAP evaluation did only show this behavior for some partition points:

- TX2+ASR, XNX+DPC, XNX+ASR

The tail model inference times on the other hand only showed this **alternating** behavior on the Desktop PC, as shown in Figures 5.3 and 5.4. This behavior was not observed for all partition points though on the Desktop PC.
Figure 5.6 is an example of how the tail model inference progressed for the different partition points on the ASRock 4x4. As shown, the tail model inference time did not show this alternating behavior, as it declined steadily with partition point depth. Although there exist two partition points, *conv_3* and *pool_3*, which shared the same tail model inference time, the time never increased when compared to the previous partition point.

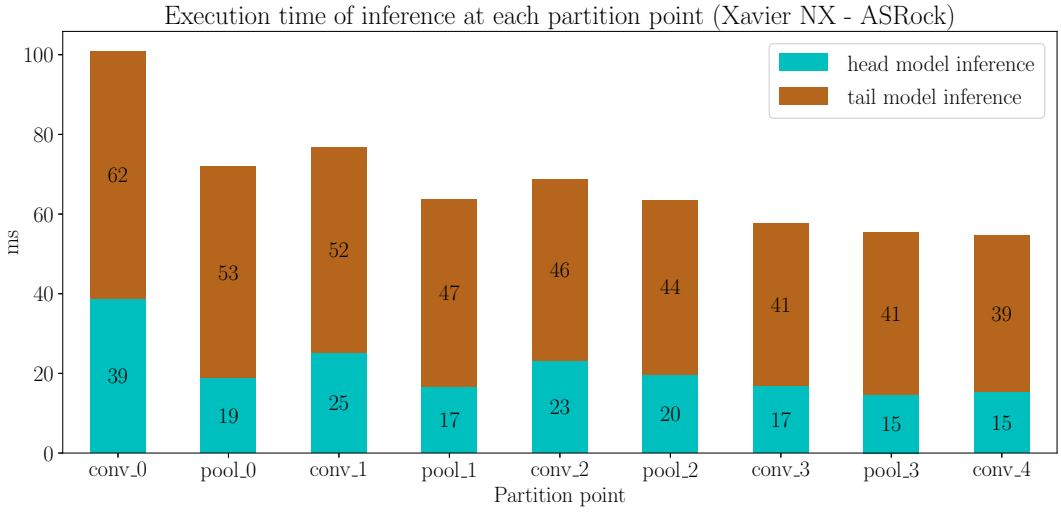

Figure 5.6: Co-Inference execution time on the Xavier NX and the ASRock 4x4

As described in Section 4.3.2 all different types of models from Table 3.3 were evaluated on the Xavier NX in addition to above benchmarks. This was done to further evaluate how the model type impacts inference of the different partition points. As already shown in Figure 5.6, the head model inference time on the Xavier NX alternately decreased and increased for some partition points, while decreasing overall with partition point

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

depth. Figure 5.7 shows the execution of the TensorFlow model on the CPU. In this case, average head model inference time increased up to partition point *conv_3*, and then declined again. When executing the same TensorFlow model on the GPU instead, which can be seen in Figure 5.8, the inference time fluctuated across all partition points, while remaining within a 8ms range.

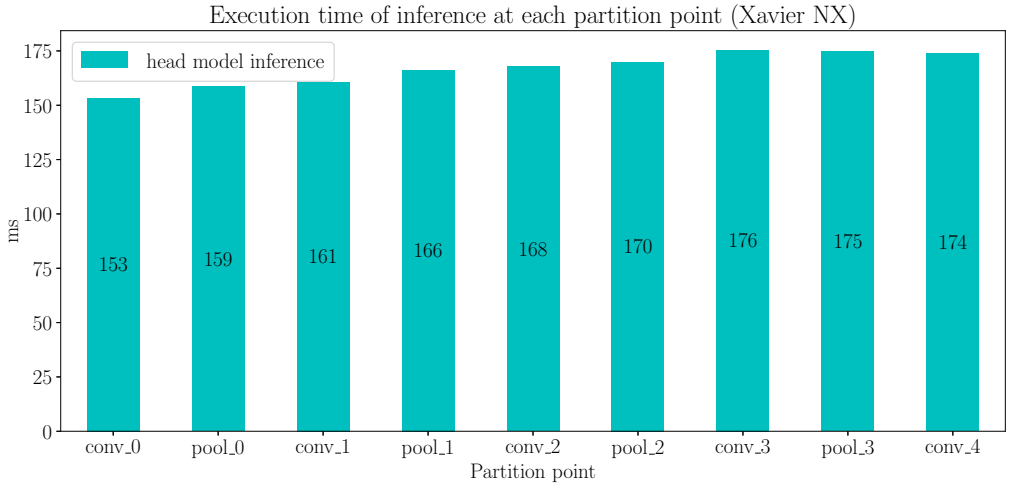

Figure 5.7: Co-Inference execution time on CPU per partition point on the Xavier NX

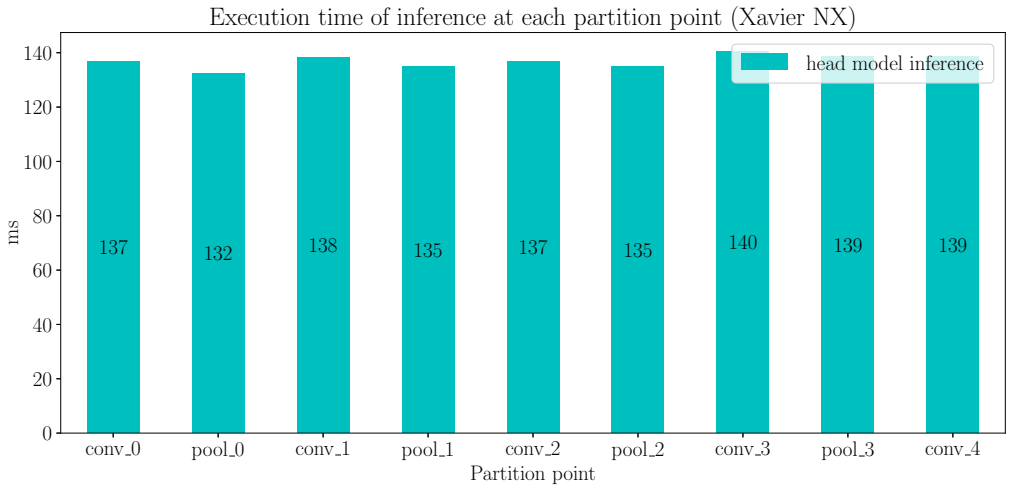

Figure 5.8: Co-Inference execution time on GPU per partition point on the Xavier NX

Figures 5.9 and 5.10 show the average head model inference time of the TensorFlow Lite and TensorFlow Lite 8-bit quantized models on the Xavier NX. Both inference times

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

showed a similar behavior, as they increased with every partition point ending on a *convolutional* layer. Additionally, some partition points ending on a max pooling layer had a lower inference time than their preceding partition points, when executing the TensorFlow Lite 8-bit quantized model. This trend is reminiscent of the head model inference time on the Raspberry Pi 3B+, as presented in Figure 5.3, which also executed the TensorFlow lite 8-bit quantized model.

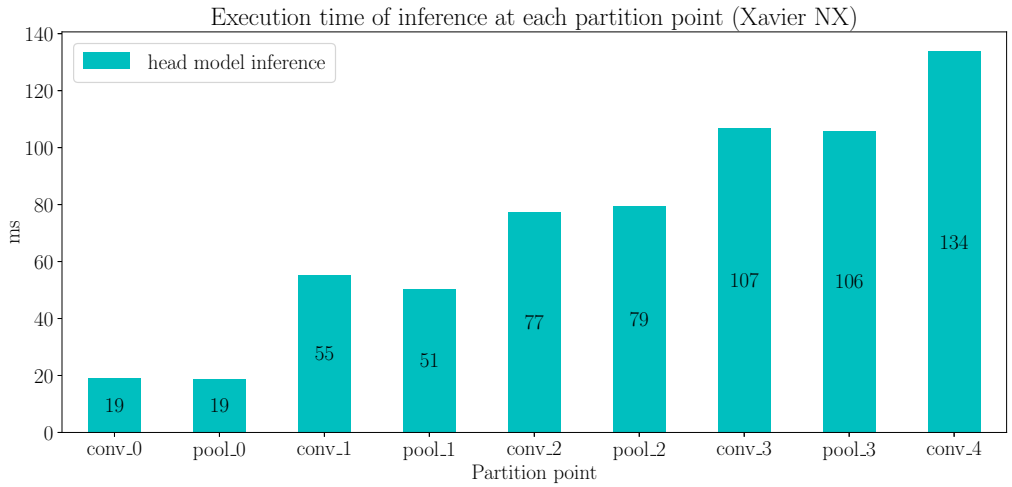

Figure 5.9: Co-Inference execution time per partition point on the Xavier NX with TensorFlow Lite

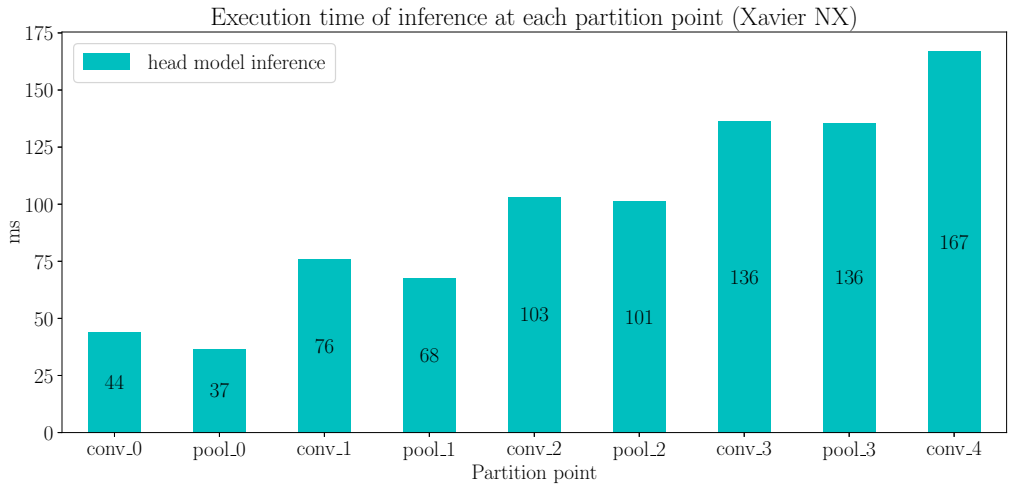

Figure 5.10: Co-Inference execution time per partition point on the Xavier NX with TensorFlow Lite 8-bit quantized

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Resource Utilization

Trends and behaviors in how the choice of partition point impacts the resource utilization of edge devices and servers during the Co-Inference VAP execution were identified. The average CPU usage of all edge devices in the Co-Inference VAP **increased** with partition point depth. This is also shown in Figure 5.11, which features the resource utilization of the Co-Inference VAP on the Raspberry Pi 3B+ and Desktop PC. In this case, the average CPU usage started at 7% for the first partition point, and increased up to 23% for the last. This increasing trend was identified in all edge devices of the Co-Inference VAP. Additionally, the CPU usage increased for each partition point ending on a *max pooling* layer, while it decreased for partition points ending on a *convolutional* layer, when compared to the preceding partition point. In some cases the CPU usage of two consecutive partition points stayed the same, which can be seen in Figure 5.11, as partition points *pool_3* and *conv_4* share the same CPU usage.

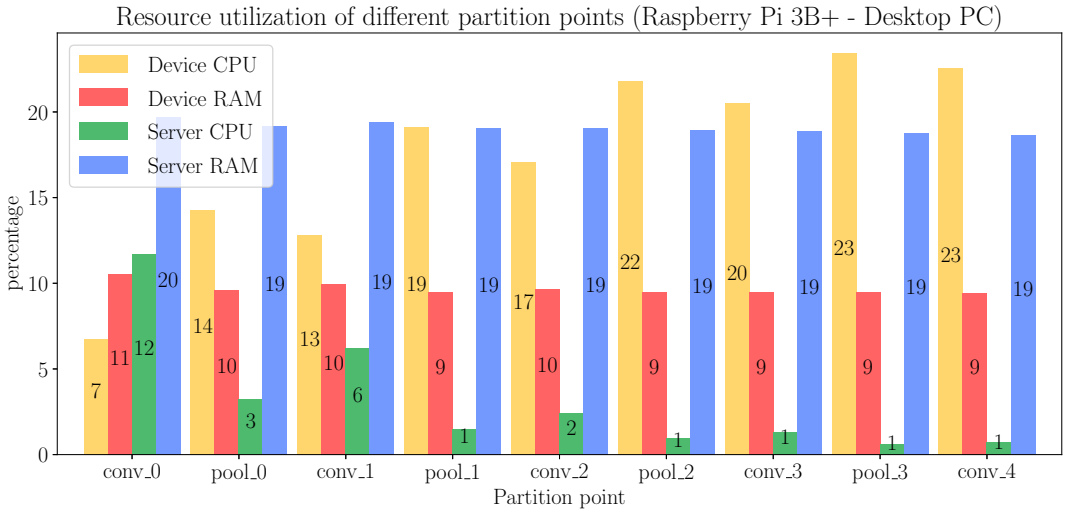

Figure 5.11: Resource utilization per partition point on the Raspberry Pi 3B+ and the Desktop PC

RAM usage of all edge devices showed a small decrease or stayed approximately the same in percentage usage for all partition points in the Co-Inference VAP. In case of the Raspberry Pi 3B+ (Figure 5.11), the first partition point had a relative RAM usage of 11%, while it was 9% for the last one. This decrease was also identified for the Tinker Edge T and Xavier NX. For the Jetson TX2 on the other hand, which is shown in Figure 5.12, RAM usage fluctuated between 26% and 27%.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

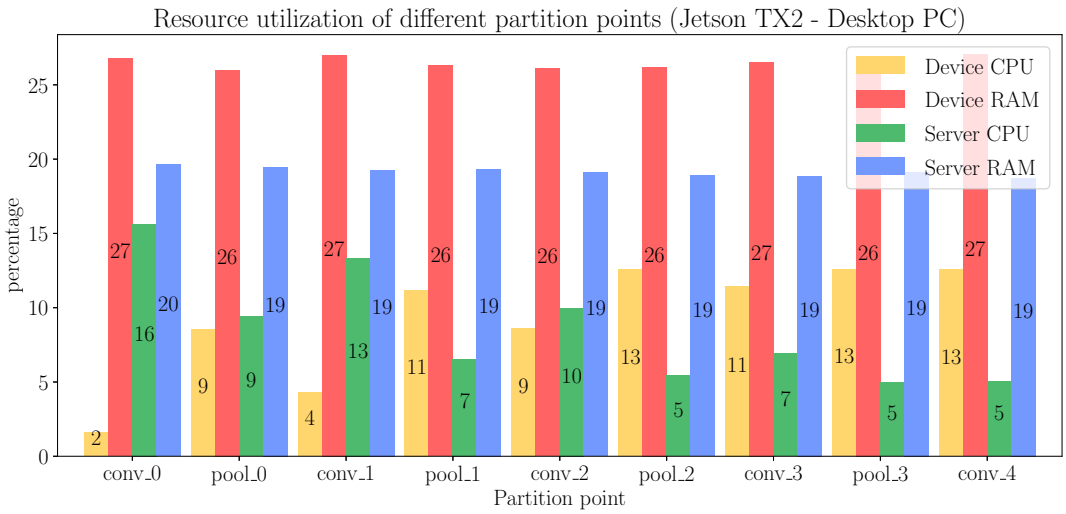

Figure 5.12: Resource utilization per partition point on the Jetson TX2 and the Desktop PC

The trend in average CPU usage differed for both edge servers. In case of the Desktop PC, CPU usage **decreased** with partition point depth, which can be seen in Figures 5.11 and 5.12. For the ASRock 4x4 on the other hand it **increased** when paired up with the Tinker Edge T, Jetson TX2 and Xavier NX, which is exemplary shown in Figure 5.13. When partaking in the Co-Inference VAP with the Raspberry Pi 3B+ instead, it **decreased** with partition point depth, which can be seen in Figure 5.14. Additionally, when comparing the CPU usage of consecutive partition points, an **alternation** similar to that of inference and CPU usage on edge devices can be observed in every Co-Inference VAP, except for the Raspberry Pi 3B+ and ASRock 4x4 pair.

As can be seen in above figures, the RAM usage of edge servers stayed approximately the same for all partition points. For the Desktop PC this usage was 19-20%, while it was 2-3% for the ASRock 4x4, for all partition points.

The rest of the resource utilization benchmarks can be found in Section A.1.2, and are referenced in below list.

- TET+DPC (Figure A.5)
- TET+ASR (Figure A.6)
- TX2+ASR (Figure A.7)
- XNX+DPC (Figure A.8)

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

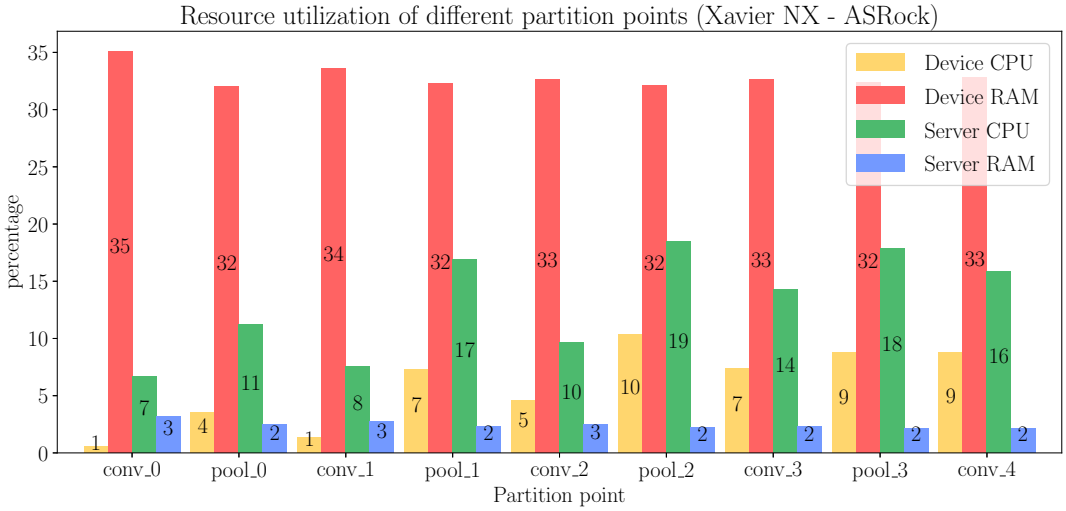

Figure 5.13: Resource utilization per partition point on the Xavier NX and the ASRock 4x4

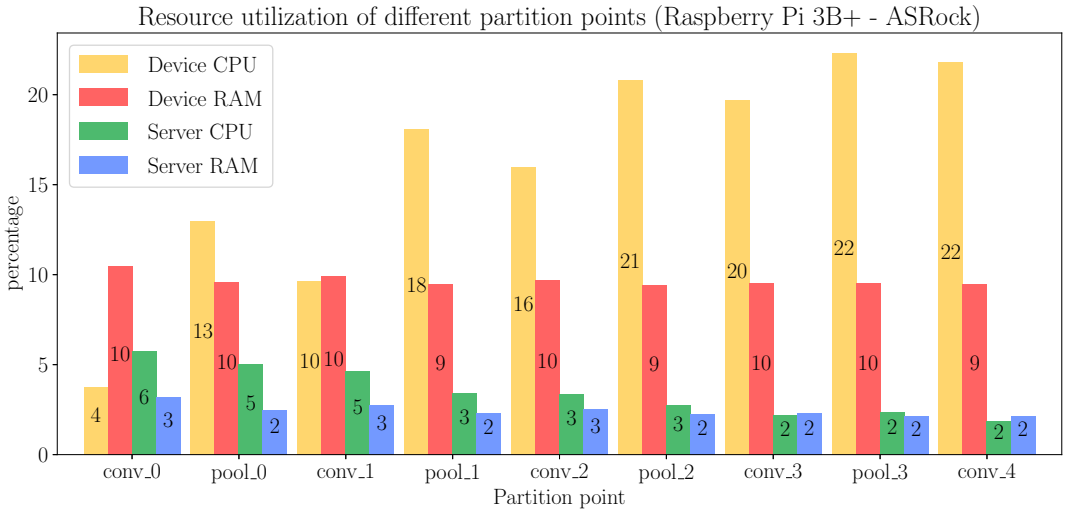

Figure 5.14: Resource utilization per partition point on the Raspberry Pi 3B+ and the ASRock 4x4

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

5.1.3 Bandwidth

The size of intermediate features of each of the partition points is shown in Figure 5.15. The size **alternately** *decreased* and *increased* when comparing consecutive partition points, starting from partition point *conv_0*. Partition points ending on a *max pooling* layer reduced the size of their input by a factor of ≈ 4, while those ending on a *convolutional* layer increased the size of their input by a factor of ≈ 2. The biggest difference can be seen when comparing *conv_0* to *pool_3*, which shows a $32\times$ reduction in size. Figure 5.16 shows the mean size of inference results, that is returned by the edge server for each frame in Bytes. This size did not vary like the size of intermediate features above, and stayed at approximately the same, having a range of 527 to 553 Bytes.

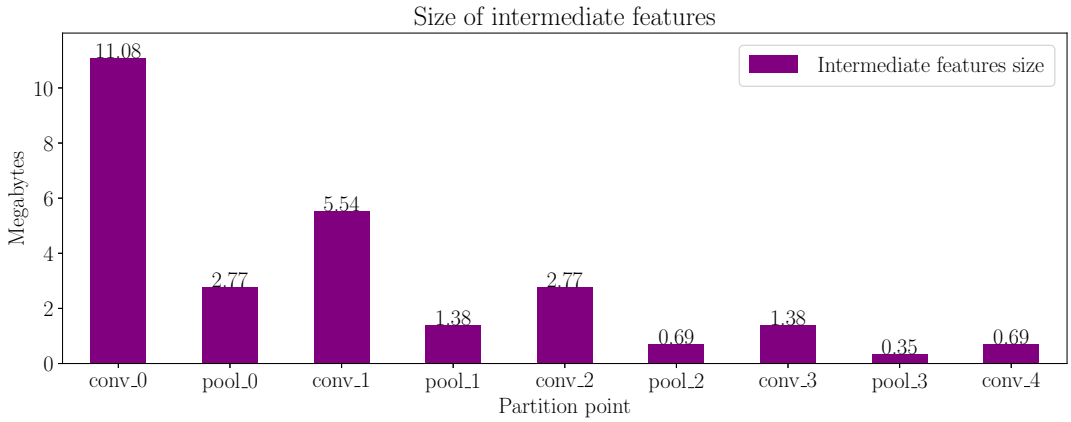

Figure 5.15: Size of intermediate features per partition point

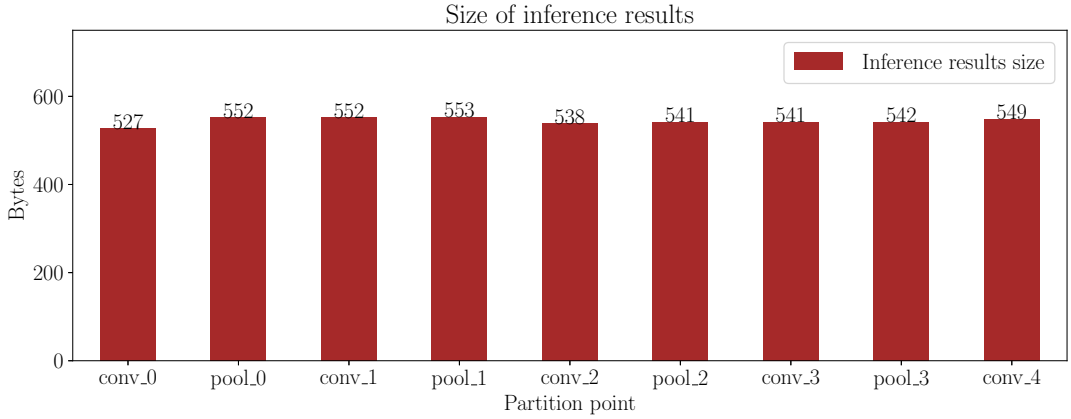

Figure 5.16: Mean size of inference results per partition point

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

5.1.4 Comparison of Local and Collaborative Performance Results

This section compares the results of the Local Inference and Co-Inference VAPs. Two subsections present the comparison of inference time and resource utilization separately.

Inference Time

Table 5.2 shows the total inference time of the Co-Inference VAP, and the available partition points, in relation to the inference time of the edge devices executing the Local Inference VAP. If the total inference time, i.e., rounded sum of head and tail model inference time of the Co-Inference VAP, is below the inference time of the edge device performing the Local Inference VAP, the cell marked as green. This Table is also presented with concrete values in Table A.1. Five device-server pairs outperformed the Local Inference VAP in terms of total inference time with at least one partition point in the Co-Inference VAP. Furthermore, total inference time in the Co-Inference VAP ranged from as low as 17% to a maximum of 337% of the local inference time. Using the Desktop PC as the edge server allowed the Co-Inference VAP to outperform the Local Inference VAP in 19 cases. For the ASRock 4x4 12 such cases were observed.

Edge Device / **Edge Server**	RBP DPC	RBP ASR	TET DPC	TET ASR	TX2 DPC	TX2 ASR	XNX DPC	XNX ASR
local	100%	100%	100%	100%	100%	100%	100%	100%
conv_0	24%	26%	145%	204%	183%	286%	217%	337%
pool_0	17%	19%	70%	125%	119%	206%	130%	240%
conv_1	28%	31%	98%	146%	144%	233%	160%	257%
pool_1	24%	26%	55%	111%	111%	194%	113%	213%
conv_2	31%	33%	61%	114%	106%	197%	143%	230%
pool_2	29%	31%	29%	98%	108%	189%	120%	213%
conv_3	34%	36%	34%	102%	100%	189%	110%	193%
pool_3	33%	35%	39%	91%	100%	167%	73%	183%
conv_4	38%	39%	29%	89%	100%	167%	63%	183%

Table 5.2: Mean inference time for each edge device in the Local Inference VAP as the baseline (100%), and relative mean total (head + tail) inference time for the device-server pairs in the Co-Inference VAP

Resource Utilization

Table 5.3 shows the percentage CPU usage of the different edge devices in the Local Inference VAP and in the Co-Inference VAP for each of the partition points. If the edge device had a higher CPU usage in the Co-Inference VAP, when compared to the CPU Usage of the edge device in the Local Inference VAP, the cell is marked as red. As the table shows, only the CPU usage for the Tinker Edge T was higher for three partition points in the Co-Inference VAP with the Desktop PC. For all other instances the CPU usage was lower in the Co-Inference VAP.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Edge Device **Edge Server**	RBP DPC	RBP ASR	TET DPC	TET ASR	TX2 DPC	TX2 ASR	XNX DPC	XNX ASR
local	25%	25%	19%	19%	18%	18%	17%	17%
conv_0	7%	4%	2%	1%	2%	1%	1%	1%
pool_0	14%	13%	11%	5%	9%	5%	8%	4%
conv_1	13%	10%	6%	2%	4%	2%	4%	1%
pool_1	19%	18%	15%	11%	11%	8%	11%	7%
conv_2	17%	16%	12%	5%	9%	4%	10%	5%
pool_2	22%	21%	20%	13%	13%	11%	14%	10%
conv_3	20%	20%	18%	11%	11%	10%	11%	7%
pool_3	23%	22%	20%	14%	13%	11%	13%	9%
conv_4	23%	22%	20%	13%	13%	11%	12%	9%

Table 5.3: CPU usage of edge devices in the Local Inference VAP and in the Co-Inference VAP for each partition point

Table 5.4 shows the percentage RAM usage of the different edge devices in the Local Inference VAP and in the Co-Inference VAP for each of the partition points. If the edge device had a higher RAM usage in the Co-Inference VAP, when compared to the RAM Usage of the edge device in the Local Inference VAP, the cell is marked as red. As shown in the table, the RAM usage was higher in five cases for the Co-Inference VAP, once for the Raspberry Pi 3B+ and four times for the Tinker Edge T. For all other instances the RAM usage was lower for all partition points in the Co-Inference VAP.

Edge Device **Edge Server**	RBP DPC	RBP ASR	TET DPC	TET ASR	TX2 DPC	TX2 ASR	XNX DPC	XNX ASR
local	10%	10%	12%	12%	36%	36%	41%	41%
conv_0	11%	10%	14%	14%	27%	27%	35%	35%
pool_0	10%	10%	11%	11%	26%	26%	32%	32%
conv_1	10%	10%	12%	13%	27%	27%	33%	34%
pool_1	9%	9%	11%	11%	26%	26%	32%	32%
conv_2	10%	10%	11%	13%	26%	26%	33%	33%
pool_2	9%	9%	11%	11%	26%	26%	32%	32%
conv_3	9%	10%	11%	11%	27%	27%	32%	33%
pool_3	9%	10%	11%	11%	26%	26%	32%	32%
conv_4	9%	9%	11%	11%	27%	27%	33%	33%

Table 5.4: RAM usage of edge devices in the Local Inference VAP and Co-Inference VAP for each partition point

Table 5.5 shows the percentage CPU usage of the different edge servers in the Local Inference VAP and in the Co-Inference VAP for each of the partition points. As the table shows, the CPU usage was lower for all partition points in the Co-Inference VAP, when comparing it to the CPU usage of the Local Inference VAP benchmarks of the edge server.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Edge Device / **Edge Server**	RBP DPC	RBP ASR	TET DPC	TET ASR	TX2 DPC	TX2 ASR	XNX DPC	XNX ASR
local	20%	35%	20%	35%	20%	35%	20%	35%
conv_0	12%	6%	15%	7%	16%	7%	16%	7%
pool_0	3%	5%	9%	11%	9%	11%	11%	11%
conv_1	6%	5%	13%	7%	13%	7%	14%	8%
pool_1	1%	3%	6%	15%	7%	15%	8%	17%
conv_2	2%	3%	9%	10%	10%	10%	9%	10%
pool_2	1%	3%	4%	17%	5%	15%	6%	19%
conv_3	1%	2%	5%	12%	7%	12%	8%	14%
pool_3	1%	2%	4%	16%	5%	16%	5%	18%
conv_4	1%	2%	4%	15%	5%	15%	5%	16%

Table 5.5: CPU usage of edge servers in the Local Inference VAP and Co-Inference VAP for each partition point

Table 5.6 shows the percentage RAM usage of the different edge servers in the Local Inference VAP and in the Co-Inference VAP for each of the different partition points. If the edge server had a higher RAM usage for a partition point, when compared to the RAM usage in the Local Inference VAP, the cell is marked as red. The RAM Usage was higher on the Desktop PC for the first partition point of the Co-Inference VAP, while it was equal or lower in all other instances.

Edge Device / **Edge Server**	RBP DPC	RBP ASR	TET DPC	TET ASR	TX2 DPC	TX2 ASR	XNX DPC	XNX ASR
local	19%	3%	19%	3%	19%	3%	19%	3%
conv_0	20%	3%	20%	3%	20%	3%	20%	3%
pool_0	19%	2%	19%	2%	19%	2%	19%	2%
conv_1	19%	3%	19%	3%	19%	3%	19%	3%
pool_1	19%	2%	19%	2%	19%	2%	19%	2%
conv_2	19%	3%	19%	3%	19%	3%	19%	3%
pool_2	19%	2%	19%	2%	19%	2%	19%	2%
conv_3	19%	2%	19%	2%	19%	2%	19%	2%
pool_3	19%	2%	19%	2%	19%	2%	19%	2%
conv_4	19%	2%	19%	2%	19%	2%	19%	2%

Table 5.6: RAM usage of edge servers in the Local Inference VAP and Co-Inference VAP for each partition point

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

5.2 Image Reconstruction Attack

This section presents the results of the image reconstruction attack, which were collected according to the methodology presented in Section 4.4. First, the original images, and the nine generated images for each partition point will be presented, to show the results of the image reconstruction attack. Then, the average values of the three different image quality metrics over all ten generated images for each partition point will be presented. Additionally, these averages are presented graphically for each of the three metrics. Additional figures show the progression of the three metrics over the duration of the image reconstruction attack, and average time needed to reconstruct an image for each partition point. At last, the singular values of the image quality metrics for each generated image and each partition point will be presented.

5.2.1 Generated Images of the Reconstruction Attack

Figures 5.17 and 5.18 show the original images *image n* used in the image reconstruction attack in the first row. The following rows feature the generated images of the partition points, which are denoted as *conv_n* or *pool_n* at the start of each row. Additionally, under the name of partition point at the start of the row, the value of D_{Batch} has been included, which is the average value of image difference over all ten generated images. As shown in both Figures 5.17 and 5.18, the generated images got subjectively worse with partition point depth. Partition points *conv_0* to *pool_1* showed near perfect reconstructions, as most details of the original image could still be made out subjectively. For *conv_2* and *pool_2* the generated images were not as recognizable as previous partition points, but an argument can still be made, that silhouettes of the original were present. Therefore, *conv_2* and *pool_2* were included in the list of partition points for which similar images were generated. The appearance of the generated images changed every second partition point, as *conv_1* introduced a green and *conv_2* a blue tint. Every generated image, for partition points after *pool_2*, showed a seemingly random constellation of pixels. Therefore, the image reconstruction attack was unable to generate subjectively similar images for partition points *conv_3*, *pool_3* and *conv_4*.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

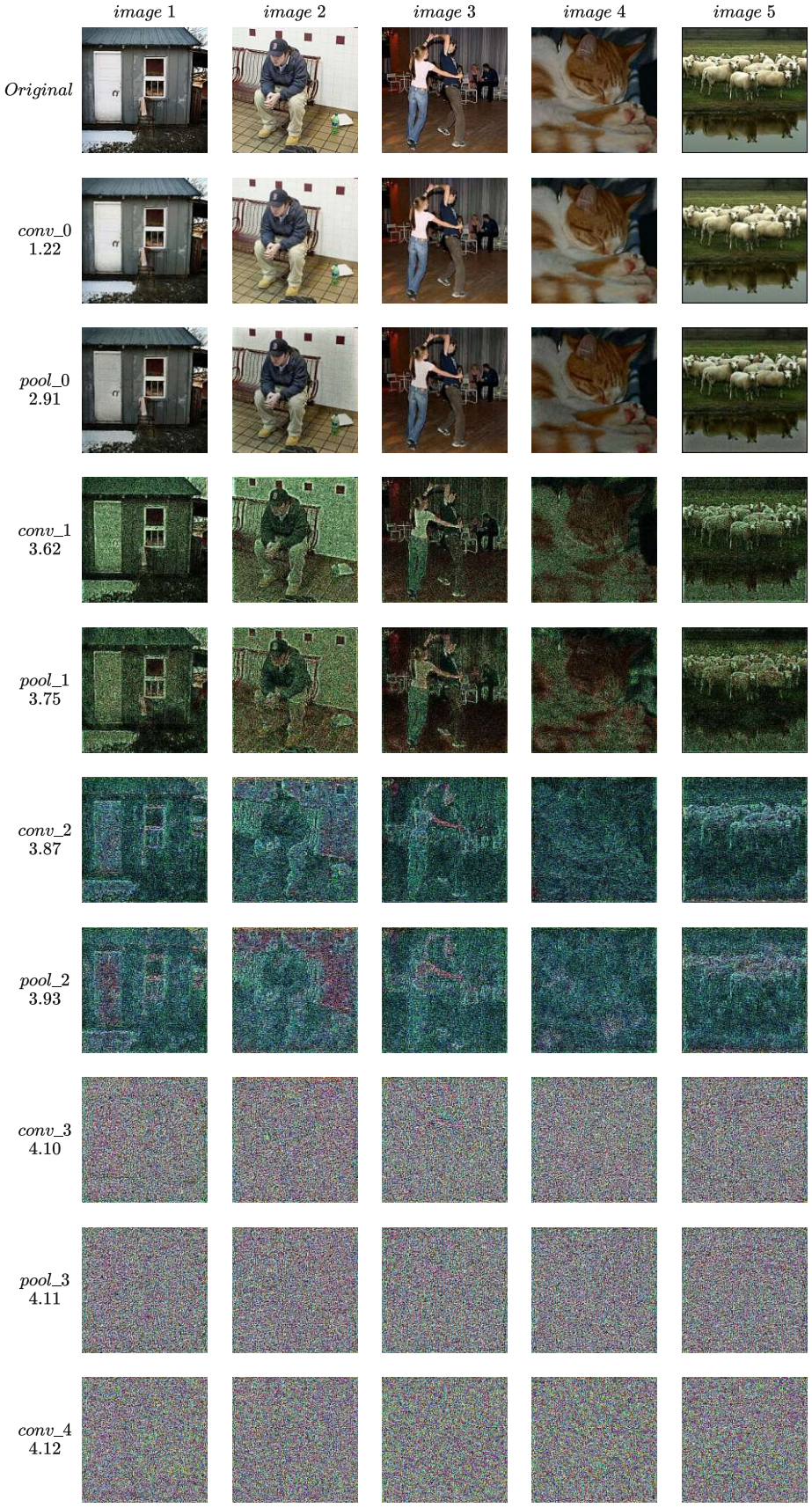

Figure 5.17: Reconstructions for images 1-5 for each partition point, including D_{Batch} over all 10 images

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

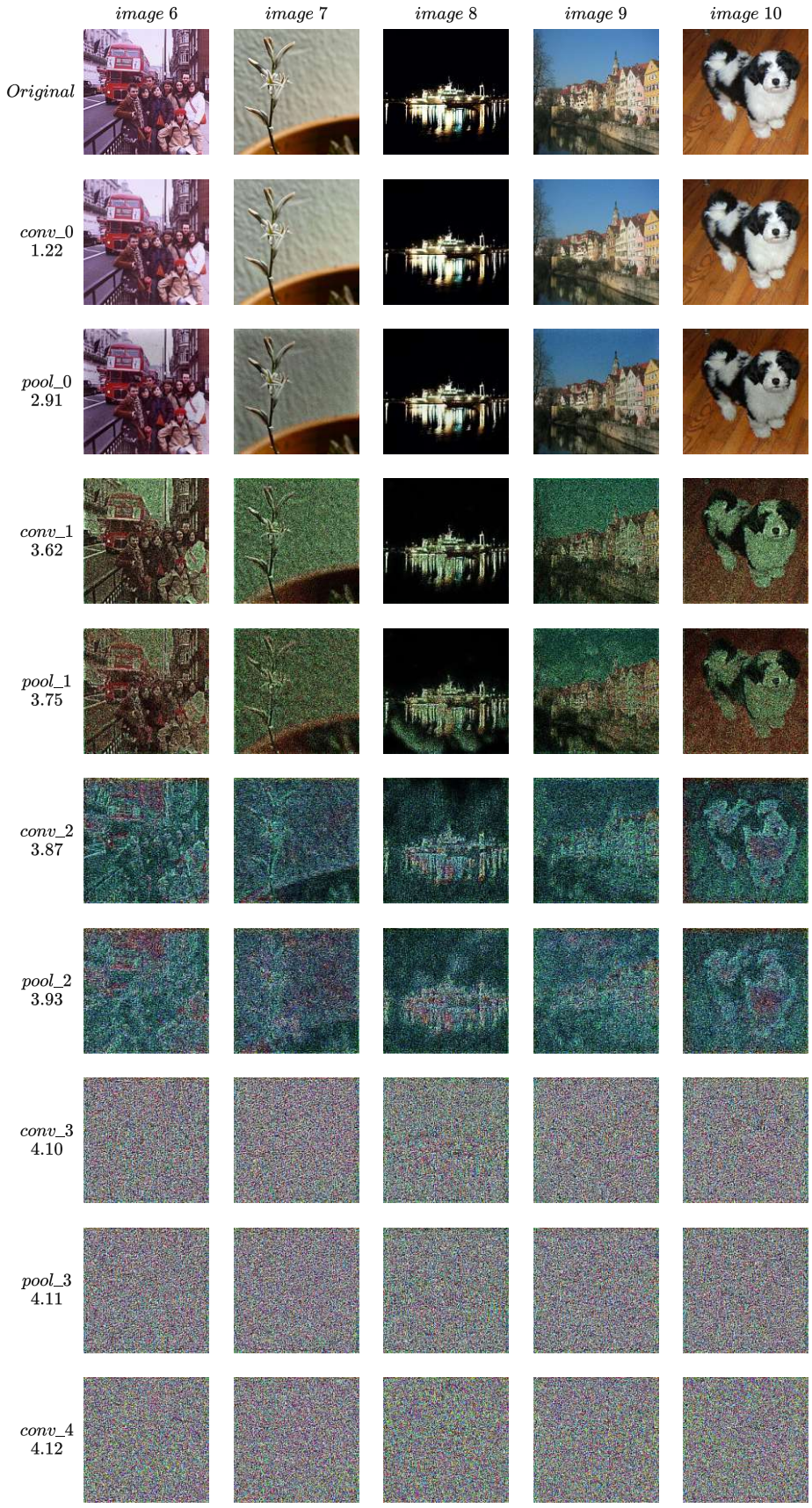

Figure 5.18: Reconstructions for images 6-10 for each partition point, including D_{Batch} over all 10 images

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

5.2.2 Average Image Quality Metrics

Table 5.7 shows the average value of the three different metrics (D, $PSNR$, $SSIM$), which were used to measure the similarity of generated and original images over all generated images, of the different partition points. Figure 5.19 shows the value of D_{Batch} for all partition points. Figure 5.20 shows the value of $mPSNR$ for all partition points. Lastly, Figure 5.21 shows the value of $mSSIM$ for all partition points. Values of D_{Batch} showed a significant jump when comparing *conv_0* to *pool_0*, as it showed an increase by a factor of ≈ 2. This observation was also identified in the values of $mPSNR$, but in this case, the comparison showed a reduction by a factor of ≈ 2. Another significant reduction was observed for $mSSIM$, when comparing *pool_0* to *conv_1*, as it was reduced by a factor of ≈ 3. For the last three partition points, the average value of D_{Batch}, $mPSNR$, and $mSSIM$, were within a range of 0.02, 0.16 and 0.0011 respectively. The difference of these metrics for *pool_2* and *conv_3* were 0.17, 1.67 and 0.0108 respectively.

Partition point	D_{Batch}	mPSNR	mSSIM
conv_0	1.22	36.21	0.9287
pool_0	2.91	19.05	0.6600
conv_1	3.62	11.91	0.2014
pool_1	3.75	10.60	0.1217
conv_2	3.87	9.48	0.0385
pool_2	3.93	8.81	0.0185
conv_3	4.10	7.14	0.0077
pool_3	4.11	7.05	0.0068
conv_4	4.12	6.98	0.0066

Table 5.7: Average image quality metrics for all images

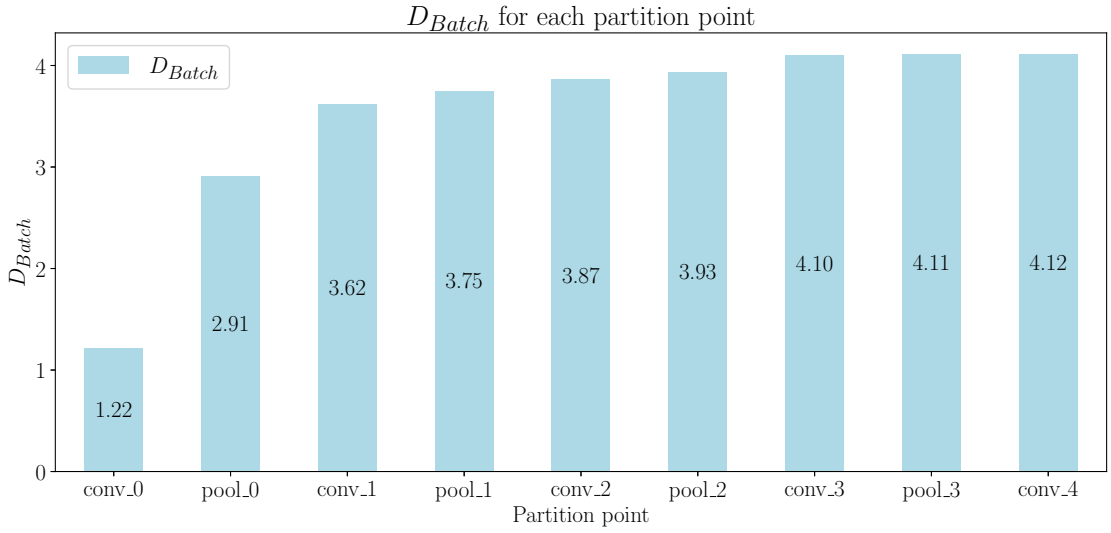

Figure 5.19: D_{Batch} for each partition point

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

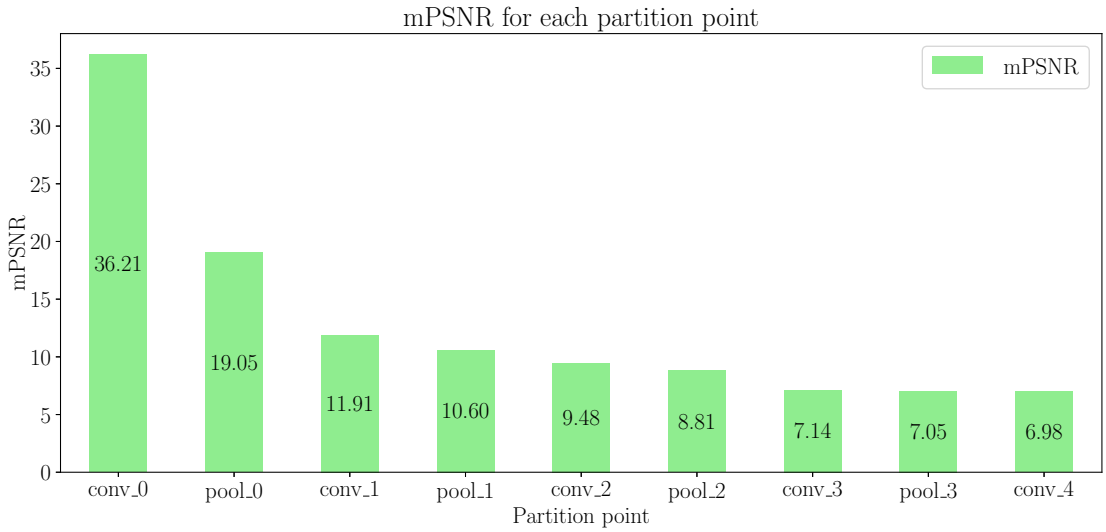

Figure 5.20: $mPSNR$ for each partition point

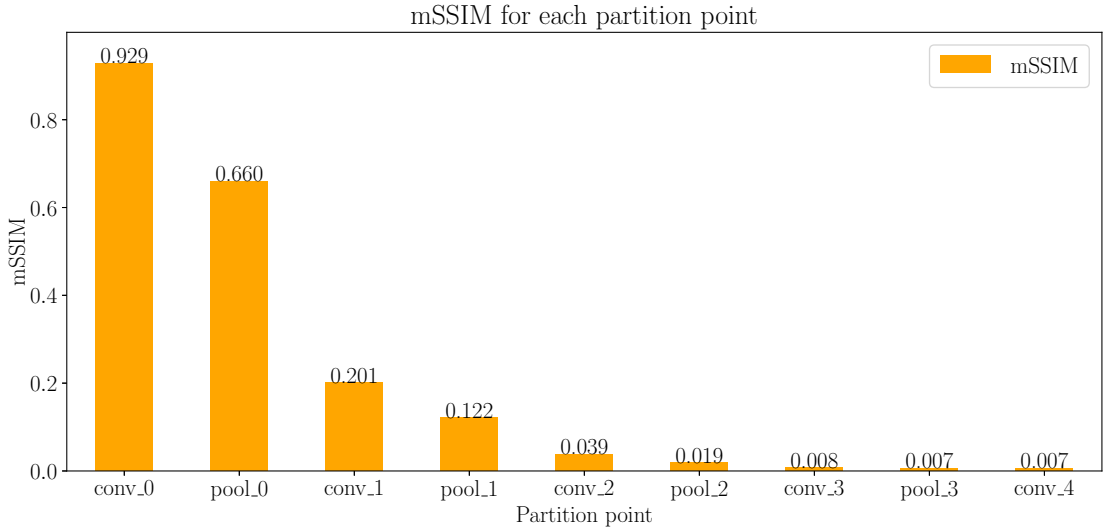

Figure 5.21: $mSSIM$ for each partition point

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

5.2.3 Attack Duration and Metric Progression over Time

This section presents the average time needed for the image reconstruction attack. Furthermore, the progression of various image similarity metrics over time is presented as well.

Average Image Reconstruction Attack Duration

Figure 5.22 shows the average time needed for reconstructing a single frame for each partition point on the GPU of the Desktop PC. Reconstructions for partition point *conv_0* took 4 seconds on average, while it took 6 seconds for the last partition point *conv_4*. Therefore, the resulting time range needed for reconstruction was 4 to 6 seconds, which gave an overall average of ≈ 5 seconds.

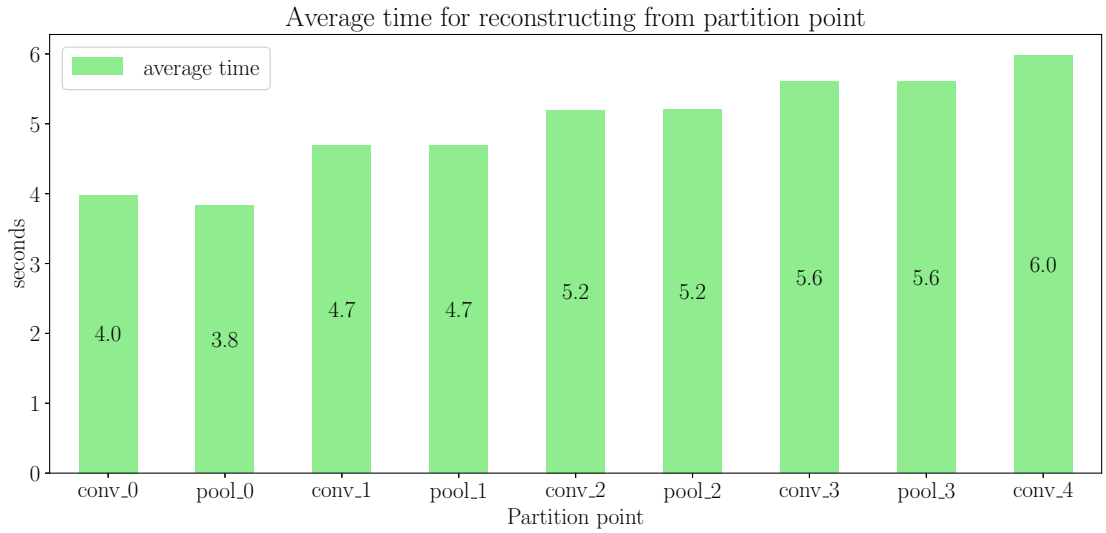

Figure 5.22: Average time needed for reconstructing from each partition point with the GTX1060 6 GB of Desktop PC

Average Similarity Metrics Progression

Figure 5.23 shows the value of D_{Batch} for each of the partition points over time, i.e., after each iteration during reconstruction. Figure 5.24 shows the same progression, but for $mPSNR$ instead. Lastly, Figure 5.25 also shows the same, but for $mSSIM$. For all three metrics, partition points *conv_0* and *pool_0* are the only ones, where the values did not appear to have stagnated after 500 iterations. The progression of all other partition points showed that after ≈ 200 iterations, values of D_{Batch}, $mPSNR$, $mSSIM$ started to stagnate. Only for partition point *pool_1* did D_{Batch} increase and $mPSNR$ decrease again after ≈ 300 iterations. For partition points *conv_1* and *pool_1* $mSSIM$ also slightly decreased after ≈ 200 iterations.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

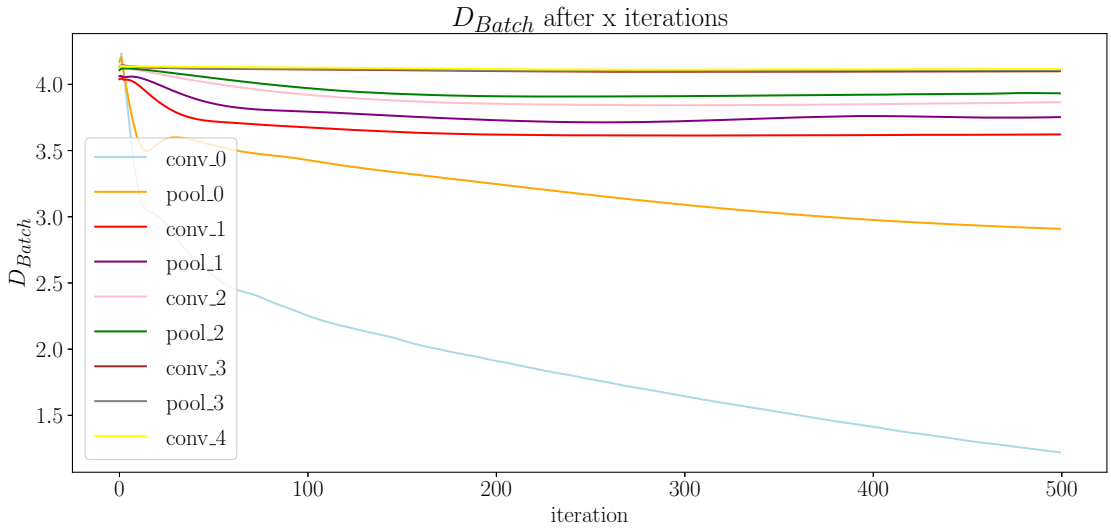

Figure 5.23: D_{Batch} after every iteration for each partition point

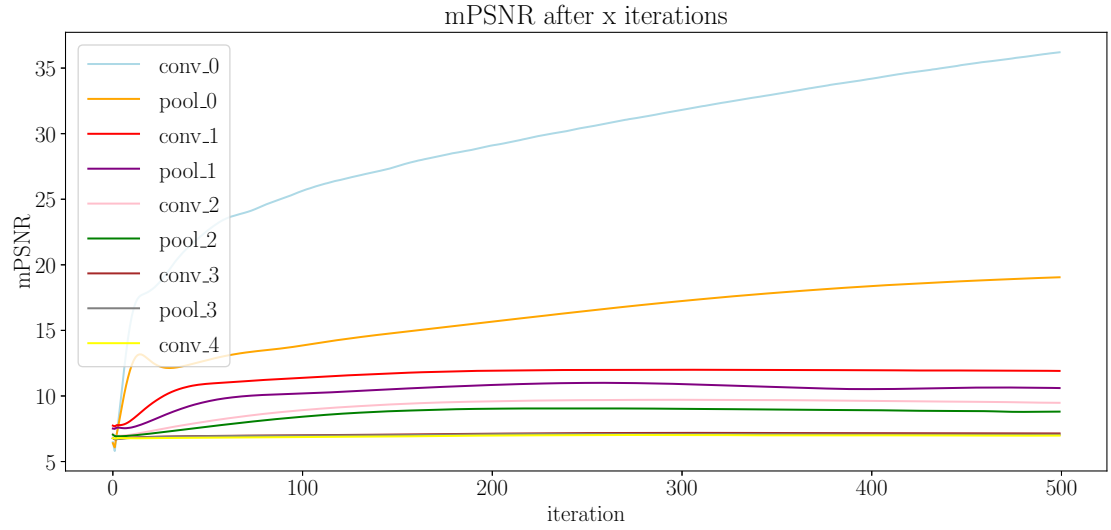

Figure 5.24: $mPSNR$ after every iteration for each partition point

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

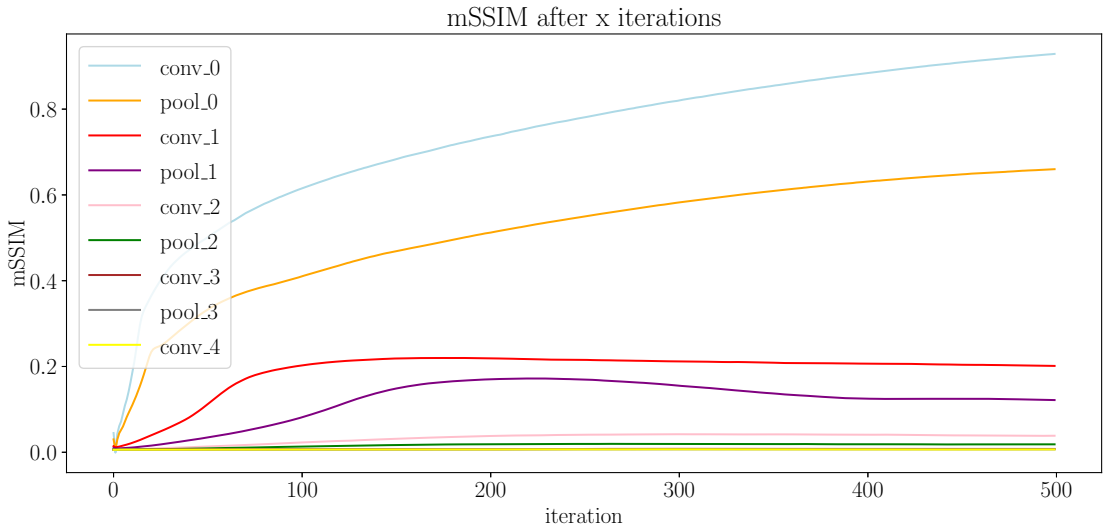

Figure 5.25: $mSSIM$ after every iteration for each partition point

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

5.2.4 All Values of Image Quality Metrics

Tables 5.8, 5.9, and 5.10, show the singular values of the three different metrics for each generated image of each partition point.

PP	image 1	image 2	image 3	image 4	image 5	image 6	image 7	image 8	image 9	image 10
conv_0	1.26	1.23	1.15	1.24	1.18	1.28	1.39	0.94	1.45	1.1
pool_0	2.99	3.23	2.6	2.91	2.86	3.11	3.33	2.0	3.02	3.04
conv_1	3.69	3.93	3.4	3.48	3.53	3.8	3.88	3.03	3.77	3.71
pool_1	3.82	4.05	3.52	3.56	3.64	3.91	3.99	3.39	3.84	3.8
conv_2	3.89	4.1	3.7	3.67	3.74	3.99	4.03	3.74	3.88	3.92
pool_2	3.95	4.12	3.81	3.76	3.85	4.02	4.03	3.9	3.91	3.98
conv_3	4.1	4.09	4.11	4.07	4.1	4.06	4.01	4.32	4.03	4.09
pool_3	4.11	4.1	4.12	4.08	4.12	4.07	4.02	4.33	4.04	4.11
conv_4	4.12	4.11	4.13	4.09	4.12	4.08	4.03	4.33	4.04	4.11

Table 5.8: Image difference D for all generated images for all partition points (PP)

PP	image 1	image 2	image 3	image 4	image 5	image 6	image 7	image 8	image 9	image 10
conv_0	35.76	36.08	37.0	36.01	36.68	35.54	34.42	39.27	33.79	37.5
pool_0	18.25	15.8	22.17	19.08	19.55	17.05	14.82	28.14	17.89	17.77
conv_1	11.22	8.79	14.13	13.36	12.87	10.18	9.29	17.79	10.43	11.03
pool_1	9.96	7.6	12.94	12.55	11.75	9.02	8.2	14.23	9.68	10.09
conv_2	9.26	7.14	11.18	11.4	10.69	8.24	7.87	10.77	9.29	8.92
pool_2	8.65	6.91	10.02	10.56	9.63	7.89	7.88	9.12	9.04	8.37
conv_3	7.11	7.24	6.99	7.43	7.11	7.51	8.01	4.96	7.87	7.18
pool_3	7.03	7.15	6.93	7.33	6.97	7.43	7.93	4.83	7.78	7.08
conv_4	6.95	7.07	6.87	7.24	6.91	7.34	7.85	4.81	7.71	7.05

Table 5.9: $PSNR$ for all generated images for all partition points (PP)

PP	image 1	image 2	image 3	image 4	image 5	image 6	image 7	image 8	image 9	image 10
conv_0	0.9419	0.9563	0.9534	0.8963	0.9363	0.9582	0.8519	0.9414	0.9004	0.9514
pool_0	0.6639	0.5957	0.734	0.6197	0.669	0.6912	0.4709	0.9011	0.6232	0.6312
conv_1	0.2007	0.1576	0.1939	0.116	0.1575	0.2152	0.0822	0.6583	0.1292	0.103
pool_1	0.102	0.0799	0.1143	0.0764	0.1037	0.1219	0.0448	0.4283	0.0736	0.0722
conv_2	0.0423	0.0401	0.043	0.0303	0.0468	0.0567	0.0255	0.0331	0.0337	0.0337
pool_2	0.0213	0.0187	0.0213	0.0175	0.0223	0.023	0.0157	0.0097	0.0187	0.0168
conv_3	0.0067	0.0089	0.0074	0.0072	0.0092	0.0107	0.0082	0.0033	0.0081	0.0074
pool_3	0.0054	0.0075	0.0082	0.007	0.007	0.0087	0.0077	0.0027	0.0075	0.0061
conv_4	0.0067	0.0069	0.0064	0.0061	0.007	0.0069	0.008	0.0023	0.0089	0.0066

Table 5.10: $SSIM$ for all generated images for all partition points (PP)

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

CHAPTER 6

Discussion and Related Work

This chapter discusses the results of the conducted experiments and compares them to related work. Performance benchmarks for throughput, resource utilization, and required bandwidth will be discussed first. Afterwards, results of the reconstruction attack will be discussed. Additionally, limitations of the conducted study will be presented. Lastly, the results and findings will be summarized.

6.1 Performance Benchmarks

This section discusses the impact, that choice of partition point has on the performance, which was measured based on throughput, resource utilization, and required bandwidth. The observed impact on each of these performance aspects is discussed separately in its own subsection. The last subsection concentrates on the comparison of Local Inference and Co-Inference VAP benchmarks.

6.1.1 Throughput

Evaluating throughput in a typical Local Inference VAP is a straight forward process, as all steps necessary to process a single frame are performed on a single device. The steps in the proposed Local Inference VAP (see Figure 3.2) are resize, inference, and post-process. To analyze a video in full, these steps have to be performed for each frame of the video. Throughput, which is usually measured in frames per second (fps), can be calculated for the presented Local Inference VAP with the following formula. Let $resize = r$, $inference = i$, and $post\text{-}process = pp$, which denote the execution times of the corresponding processing step in milliseconds.

$$fps = \frac{1000}{r + i + pp} \tag{6.1}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

As transmission of intermediate features and inference results are also processing steps of a Co-Inference VAP, which was shown in Figure 3.3, these transmission timings have to be considered for throughput in Co-Inference as well. Furthermore, as inference is split up between two devices, the head model and tail model inference times have to be measured on the two devices. Let $head\ model\ inference = h$, $transmission\ to\ server = tts$, $tail\ model\ inference = t$, $transmission\ to\ edge\ device = tte$, which once again denote the execution times of the corresponding processing step in milliseconds. Then throughput for a Co-Inference VAP can be calculated with the following formula.

$$fps = \frac{1000}{r + h + tts + t + tte + p} \tag{6.2}$$

Latency is heavily affected by environmental factors, such as signal strength, signal quality, interferences, or simply heterogeneous hardware infrastructures like cable type, networking devices, and many more. Hence, measuring latency parameters would not be helpful to demonstrate the approach presented in this thesis, and is therefore out of scope, as previously mentioned in Section 4.3. Therefore, *tts* and *tte* are not part of this discussion, and it will focus on the impact Co-Inference and different partition points can have on the inference time, i.e., h and t, instead.

Nevertheless, transmission of intermediate features are mandatory in Co-Inference VAPs, which is why the bandwidth requirements will be discussed in a later section. Co-Inference further has no impact on both the resize and post-processing step, as these are only dependent on computing power of the edge device, which further removes them from the discussion material.

6.1.2 Inference

This section discusses how inference time was impacted by the various partition points of the implemented Co-Inference VAP. The subsections present different observations, which are based on the results presented in Section 5.1.2.

Alternating Inference Time

Intuitively, the time needed to finish inference on any device, in a Local Inference or a Co-Inference VAP, should increase with each additional layer executed, as more work should lead to longer processing time. But this is not the case, as highlighted by the Co-Inference results in Section 5.1.2. Instead, it has been observed, that in most cases, the head model inference time alternately decreased and increased for each additional layer included in the partition point. This fluctuation in inference time is related to the partition points alternately ending on a *convolutional* or *max pooling* layer. Max pooling layers quarter the output of the previous *convolutional* layer, as described in Section 2.2.5. This can also be observed in Figure 5.15, which presents the size of intermediate features for each partition point. It shows, that the output of each partition point ending on a *convolutional* layer (conv_0, conv_1, conv_2, conv_3, conv_4) had 4 times the size of the following partition point ending on a *max pooling* layer (pool_0, pool_1, pool_2,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

pool_3). Therefore, it can be argued, that the inference time is also dependent on how big the output of a certain partition point is. This could be due to writing back results from the machine learning model back into memory taking up substantial time, but was not confirmed within the scope of this thesis. These results suggest, that when given the choice between a partition point ending on a *convolutional* layer or the following *max pooling* layer, the *max pooling* layer should achieve a lower inference time and should therefore be prioritized.

Kang et al. [KHG+17] also analyzed the execution time of AlexNet at each partition point, shown in Figure 6.1, which matches the evaluation methodology of this thesis. They also include the latency inbound through data communication, which was not dealt with in this thesis. Because no concrete values are given for inference times of single partition points, and their graphs also included no numerical values, it is hard to confirm if their results showed the alternating behavior which was determined for the partition points of YoloV3-Tiny.

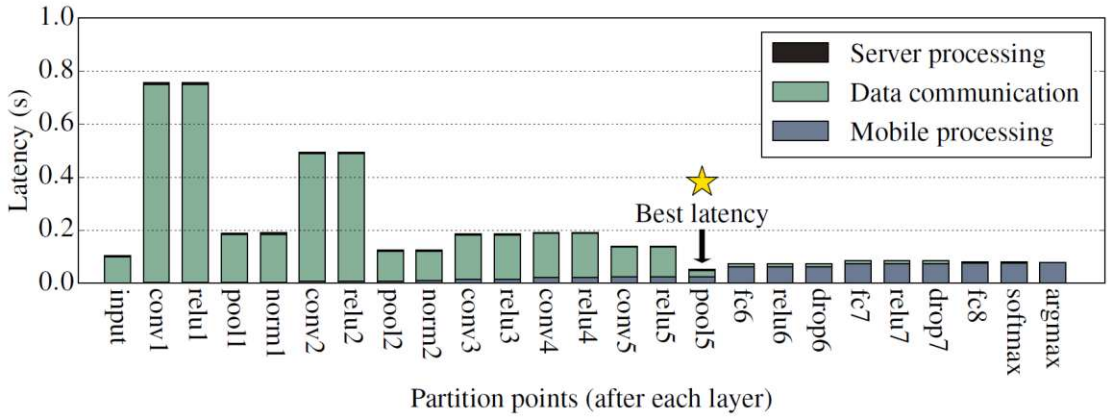

Figure 6.1: Latency of AlexNet partition points, by Kang et al. [KHG+17]

Execution Time of Model Types

Another important discovery were the two different trends that were identified when looking at the head model inference times of all partition points. These two trends also directly impacted the total inference time, for which these trends were also identified. For the Raspberry Pi 3B+, the results (Figures 5.3 and A.1) showed the increasing trend, i.e., time to finish the head model inference increased with partition point depth. For the other three edge devices, the results (Figures 5.4, A.2, A.4, 5.6, A.3, and 5.5) showed the decreasing trend. As this behavior was suspected to be caused by the used model type, i.e., which version of the model was used for inference, all model types presented in Table 3.3 were evaluated on the Xavier NX. Resulting benchmarks were already presented in Section 5.1.2.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

This analysis seems to confirm the initial suspicion, that the model type impacts how inference time of various partition points will differ from each other. While inference time decreased for the quantized, accelerator specific models with partition point depth, it increased for all other model types, and stayed approximately the same when executed on the GPU. Average inference time for the Xavier NX was 132-140 ms for the TensorFlow model on the GPU (see Figure 5.8), while it was only 13-39 ms (see Figures A.4 and 5.6) for the TF-TRT 8-bit quantized model, which also runs on the GPU. This confirms that the converters, which carried out quantization of the TF-TRT and Edge TPU models, optimize the models in such a way, that allow the devices to perform inference with these models in a fraction of the original inference time. Furthermore, it seems counterintuitive, that inference time declined with partition point depth for the TF-TRT 8-bit quantized and Edge TPU model. This further supports the theory, that inference time is affected by intermediate feature output size, which was already mentioned in above section. When comparing the inference time of the Tinker Edge T (Figure 5.4) to the size of intermediate features of each partition point (Figure 5.15), it seems that these two are directly related, as both figures mirror each other in increase and decrease. If after each inference execution, the intermediate features have to be transferred from the memory of the GPU or accelerator to the main memory, the required time for this operation could lead to the observed behavior. As above theory was not validated within this thesis, low-level profiling would be needed, to gather further insights.

Above insights can further be used to predict the behavior the choice of partition point can have in a Co-Inference VAP. For example, if an edge device has no accelerator, TensorFlow Lite models (both quantized and non quantized) could offer faster inference speeds for shallower partition points than TensorFlow models, as suggested by the results of both TensorFlow Lite models on the Xavier NX and Raspberry Pi 3B+.

Kang et al. [KHG+17] also measured the execution time of machine learning models at each partition point, and perform inference on the GPU of a Jetson TK1. In their analysis of AlexNet, see Figure 6.1, mobile processing increases with partition point depth. In the case of GPU execution on the Xavier NX, see Figure 5.8, the inference time stayed approximately the same for all partition points.

Tail Model Inference Time

The tail model inference time decreased with partition point depth on both edge servers. However, the alternating behavior regarding tail model inference time, as described in Section 6.1.2, was also observed on the Desktop PC. This may again be due to the different model types, as the Desktop PC used the TF-TRT 8-bit quantized model, while the ASRock 4x4 performed inference with the TensorFlow model. However, further investigation is needed to confirm this theory, which is part of future work.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

6.1.3 Resource Utilization

The different partition points of the Co-Inference VAP and their resource utilization were presented in Section 5.1.2. As indicated by the results, Co-Inference and choice of partition point do not significantly impact RAM usage on edge devices and servers. The CPU Usage increased with partition point depth for edge devices, which suggest that if the main goal is to alleviate CPU usage on edge devices, a shallower partition point may help in achieving that goal. Additionally, an alternating CPU usage pattern was observed, which is similar to that of inference times. Concretely, the CPU usage increased for partition points ending on *max pooling* layers, while it decreased for those ending on *convolutional* layers, when comparing the CPU usage to the preceding partition point. These results suggest that a partition point ending on a *convolutional* layer should be chosen to reduce CPU usage on edge devices. This further suggests, that there exists a trade-off between CPU usage and inference time, when choosing a partition point. This is due to inference time decreasing with partition points ending on *max pooling* layers, as described in Section 6.1.2, while CPU usage increases. Based on the requirements of a Co-Inference system, e.g., if resource utilization is prioritized over inference speed, an adequate partition point can be selected based on these results.

6.1.4 Bandwidth

This section discusses how the required bandwidth of the Co-Inference VAP was impacted by the different partition points. Furthermore, the impact these bandwidth requirements may have on choice of partition point are presented.

Size of Intermediate Features

Size of intermediate features, which was presented in Section 5.1.3, differed in their size based on partition point. This was already expected, as Figure 3.5 also shows the variety in output dimensions of the various layers within YoloV3-Tiny. Deeper partition points are therefore better suited to minimize bandwidth requirements, as shown by both the evaluation results in Section 5.1.3 and Figure 3.5.

Additionally, the size of intermediate features, alternately decreased and increased when comparing consecutive partition points. This alternation, as already explained in Section 6.1.2, was due to the partition points alternately ending with *max pooling* and *convolutional* layers. These results suggest, that when given the choice between a partition point ending on a *convolutional* layer or *max pooling* layer, the latter should be prioritized to reduce bandwidth requirements in a Co-Inference VAP. Furthermore, bandwidth requirements can be predicted, by looking at architecture specifications of a machine learning model, such as Figure 3.5.

Li et al. [LZC18] and Kang et al. [KHG+17] performed similar bandwidth analysis for multiple networks, including AlexNet and VGG. These networks also featured a similar alternation of *convolutional* and *max pooling* layer as YoloV3-Tiny. In their work they made two concrete observations which were confirmed within this work. First,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

intermediate features size decreased with partition point depth of their evaluated models. Second, an alternating decrease and increase in intermediate feature size was identified, caused by the alternation of *max pooling* and *convolutional* layers. This analysis was already shown in Figure 2.10 for Li et al. [LZC18] and is shown in Figure 6.2 for Kang et al. [KHG+17].

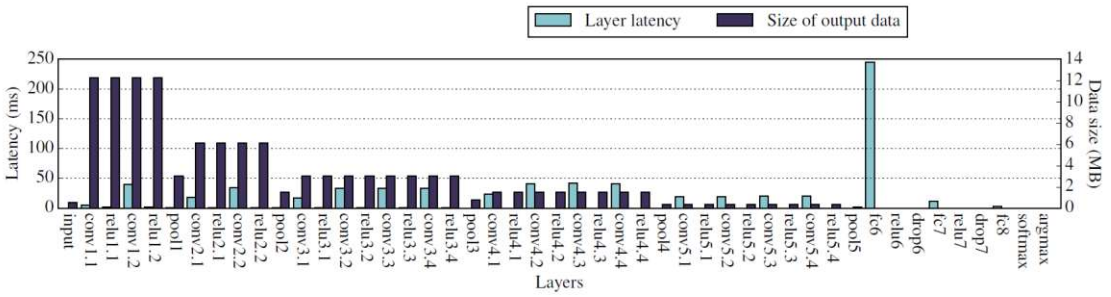

Figure 6.2: Per layer execution time and size of intermediate features for VGG, by Kang et al. [KHG+17]

Size of Inference Results

Edge servers completed inference with received intermediate features, and returned the inference results back to the edge devices. As presented in Section 5.1.3, the size of these inference results was between 527 and 553 Bytes, showing a relatively small fluctuation in size. This can be explained, as tail models always end on the same layer, regardless of partition point. Therefore, as only one sample video was used for the evaluation of the Co-Inference VAP, the inference should produce approximately identical results.

The bandwidth required for transmitting inference results was in the size of Bytes, while intermediate features were in the size of Megabytes. Therefore, the size of inference results is negligible when compared to the size of intermediate features. Even at the second to last partition point (*pool_3*), which had the lowest bandwidth requirements, 350 Kilobytes had to be transferred from the edge device to the edge server for each frame on average. Comparing these 350 KB to the average of ≈ 0.55 KB for the inference results clearly show that most of the bandwidth requirements stem from the transmission of intermediate features.

Choice of Partition Point

The results of above analysis strongly suggest, that choice of partition point can greatly impact the bandwidth requirement of a Co-Inference VAP. Therefore, to keep these requirements as low as possible, a partition point with small intermediate feature size should be chosen in a production-grade Co-Inference VAP. This thesis ignored transmission time of intermediate features, and latency, as explained in Section 6.1.1. Nevertheless,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

bandwidth requirements can directly impact transmission time. Hence, partition points resulting in low intermediate feature size, will also help to reduce transmission time.

6.1.5 Comparison of Local and Collaborative Performance Benchmarks

Previous sections focused on the benchmarks of the Co-Inference VAP, and how different partition points impact various performance aspects. As these performance benchmarks were also collected for the Local Inference VAP, their comparison is discussed in this section.

Inference

The Local Inference VAP executed on each edge device was benchmarked according to the methodology presented in Section 4.3. Additionally, each of these edge devices executed the Co-Inference VAP with two different edge servers, of which the benchmarks were also recorded. This section compares the benchmarks of the Local Inference VAP on a specific edge device to those of the Co-Inference VAP, in which the same edge device took part. The results of this comparison was presented in Section 5.1.4. The variety of the results is condensed and discussed in this section.

Five device-server pairs were able to outperform the Local Inference VAP in terms of total inference time with at least one partition point when executing the Co-Inference VAP. For the Raspberry Pi 3B+, all partition points yielded quicker inference, i.e., less total inference time with both edge servers. The Co-Inference VAP consisting of the Tinker Edge T and Desktop PC also yielded a quicker inference with all partition points, except for the first partition point. In contrast, pairing up the Tinker Edge T and ASRock 4x4 resulted in quicker inference with only three partition points. For the Jetson TX2 both evaluations of the Co-Inference VAP resulted in a higher total inference time for all partition points. Similar results were observed for the Xavier NX, except for two partition points when partaking in Co-Inference with the Desktop PC, where total inference time was once again lower. In some instances, the execution of only the head model on the edge device was even slower, than executing the Local Inference VAP on the edge device. For example, both Nvidia devices achieved a lower inference time in the Local Inference VAP, than when executing the head model up to the first partition point in the Co-Inference VAP. To summarize, utilizing the Desktop PC as the edge server in the Co-Inference VAP resulted in 19 cases where it outperformed the Local Inference VAP. For the ASRock 4x4 only 12 such cases were observed.

The variety of these results shows, that while achieving lower total inference time with Co-Inference is possible, it is very dependent on the devices participating in it. The results suggest, that low-end edge devices, such as the Raspberry Pi 3B+, benefit the most from Co-Inference. This is further supported by Table 5.2. It shows the total inference time of each device-server pair executing the Co-Inference VAP, in relation to the total inference time of the Local Inference VAP on the edge device of each Co-Inference VAP. For example, the Co-Inference VAP on the Desktop PC and Raspberry Pi 3B+ achieved

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

a total inference time that was only 17-38% of the total inference time of the Local Inference VAP on the Raspberry Pi 3B+. Even when the Raspberry Pi 3B+ was paired up with the ASRock 4x4, an edge server without an accelerator, the Co-Inference VAP achieved a total inference times as low as 19-39% of the Local Inference VAP. On the other hand, high-end edge devices partaking in the Co-Inference VAP, such as the Xavier NX and Jetson TX2, failed to outperform the Local Inference VAP of the edge devices, except for two partition points. In the case of the Xavier NX and ASRock 4x4, the total inference time was more than three times as high for the first partition point, when compared to the Local Inference VAP on the Xavier NX alone. Using the Tinker Edge T in the Co-Inference VAP shows, how the edge server can either lead to lower or higher total inference time. For the Tinker Edge T, eight partition points yielded a lower total inference time when paired up with the Desktop PC. When paired up with the ASRock 4x4 instead, only three partition points yielded a lower inference time.

The comparison of inference times of the Local Inference and Co-Inference VAP ignored transmission of intermediate features, as mentioned in Section 6.1.1. Throughput would be negatively impacted when considering this transmission aspect. The Tinker Edge T showed marginal improvement in the Co-Inference VAP with the ASRock 4x4, as the three partition points that outperformed the Local Inference VAP achieved 98%, 91%, and 89% proportional total inference time of the Local Inference VAP. For these partition points, the additional transmission would probably already lead to a lower throughput.

Resource Utilization

The resource-utilization benchmarks of the Co-Inference VAP were compared to those of the Local Inference VAP in Section 5.1.4. The results of this comparison suggest, that CPU usage can be reduced on edge devices, when participating in a Co-Inference VAP instead of performing inference on the edge device alone. They also suggest, that an edge server partaking in Co-Inference has a significantly lower CPU usage, than when inference is only performed on the edge server alone. This may be due to the edge server spending some time in idle during Co-Inference, while waiting for the next batch of intermediate features. Furthermore, RAM usage on edge servers in the Co-Inference VAP did not significantly change from the RAM usage of the Local Inference VAP on the edge servers, as shown by the results in Table 5.6. They further suggest, that RAM usage on edge devices is only significantly lower for NVIDIA devices performing Co-Inference, when compared to the RAM usage of the edge device performing inference locally, as shown by the results in Table 5.4.

Bandwidth

Bandwidth needs of the partition points of the Co-Inference VAP can not be compared to those of the Local Inference VAP, as the latter processes all data locally, without the need of transmission. The bandwidth requirements can be compared to that of a typical offloading VAP though, in which the video is sent to the processing server frame by frame. In the case of the test video, of which the properties were outlined in Table 3.1,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

0.6 Megabytes have to be transmitted for each frame. This size is due to framework specific transmission characteristics of the evaluation prototype. Comparing 0.6 to 0.35 Megabytes of partition point *pool_3* in Figure 5.15 shows, that in this case, Co-Inference had lower bandwidth requirements than typical offloading.

6.2 Image Reconstruction Attack and Impact on Data Protection

Offloading video data to powerful servers is a popular method to enable Edge Intelligence, i.e., to analyze videos from edge devices with machine learning models. In the case of offloading though, the receiving server could act maliciously. Even if the transmission is encrypted, the receiving server would have to decrypt the transmission for inference, which allows him to obtain the original video data. This entails privacy concerns, as the receiving server could process the video unlawfully.

Another method to facilitate Edge Intelligence is Collaborative Inference, in which two devices perform inference together. For that, the original machine learning model has to be split into two models. This requires the selection of a partition point, i.e., after which layer the model is split up. To mitigate above privacy concerns, RQ_2 deals with the question of how different partition points impact the privacy of a Co-Inference VAP.

In Co-Inference, the transmission of intermediate features can mitigate above privacy risks. Even if an adversary gets access to intermediate features, sophisticated methods are necessary to view the original image data. An image reconstruction attack is a typical example of such a sophisticated method and was carried out within this thesis. This attack takes intermediate features as its input and aims to reconstruct the original image data, as explained in more detail in Section 3.5.2.

In general, image reconstruction attacks can be categorized into three different categories, as defined by He et al. [HZL19]. These categories include white box, black box, and query-free attacks, which differ in their requirements. In this thesis a white box attack was used to analyze how different partition points impact the privacy of the Co-Inference VAP, specifically the attack as described in [OL19].

Additionally, the paper describing the applied attack, also introduces a metric to calculate how similar two images are, called image difference (D). Other image similarity metrics include SSIM, proposed by [WBSS04], and PSNR. Structural Similarity Index (SSIM) and Peak Signal-to-Noise Ratio (PSNR) are also commonly used to measure the similarity of two images. In this thesis, all three metrics (D, SSIM, PSNR) were implemented to measure the similarity of generated to original image. This methodology allows for independent and scientifically sound evaluation results, thus a far more accurate proposal for adequate partition point selection towards increasing privacy can be made.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

6.2.1 Image Similarity and Impact on Privacy

The nine possible partition points of the Co-Inference VAP, which were evaluated for performance, were also subjected to the image reconstruction attack as described above and explained in Section 3.5.2. Selecting a particular partition point in a Co-Inference VAP shows, that this selection has a significant impact on privacy, as can be seen in Figures 5.17 and 5.18.

Privacy Preserving Partition Points

Personally identifiable information (PII), per the definition of the General Data Protection Regulation (GDPR) (see Section 2.4), *'means any information relating to an identifiable natural person'* [Cou22]. Analyzing the results of both Figures 5.17 and 5.18 for PII subjectively, i.e., with human perception, suggests, that for all generated images of partition points *conv_3*, *pool_3*, and *conv_4*, no PII are present as nothing can be identified by looking at the generated images. This strongly suggests that a Co-Inference VAP using these partition points would be able to withstand the applied image reconstruction attack, and adhere to GDPR mandates in regard to privacy.

However, is has to be noted that there is no guarantee that generating images for these three partition points will never expose PII. Therefore, generated images still have to be subjected to human perception, to verify if PII are exposed. Hence, above discussion made its statements based on human perception, i.e., they were subjectively assessed, to formulate findings based on the generated images.

Objective Similarity Metrics

Objectively measuring the quality of reconstruction, i.e., if a generated image exposes PII, was conducted with the similarity metrics presented in Section 3.5.3. The origin of D, image difference, lies within the paper of Oh et al. [OL19], which also proposed the image reconstruction attack. The other two metrics, $SSIM$ and $PSNR$, were originally found in the works of He et al. [HZL19], but were not invented by them. The results of Oh et al. [OL19] suggested, that their metric is a valid way of measuring the similarity of reconstructed to an original image. Figure 6.3 shows the quality of reconstruction one may expect for a specific image difference value. Images with an image difference of up to 2, should show (subjectively) similar images, while values above 3 start to distort the images. Lastly, anything above 4 starts to be unrecognizable, which they also suggest as a possible threshold of when the image reconstruction attack leads to no usable results.

Figure 5.19 shows the mean value of image difference over all 10 generated images for each partition point. For example, the mean image difference for partition point *conv_0* was 1.22, 2.91 for *pool_0*, 3.62 for *conv_1*, and so on. These values and the reconstructed images do not line up with those of Figure 6.3. Discussing this mismatch is even easier, as two of the images subjected to the image reconstruction were intentionally taken from the paper. The first image features the city of Tübingen (image 9), while the second one features a picture of a dog (image 10), more specifically a Tibetan terrier. Figure 5.18

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

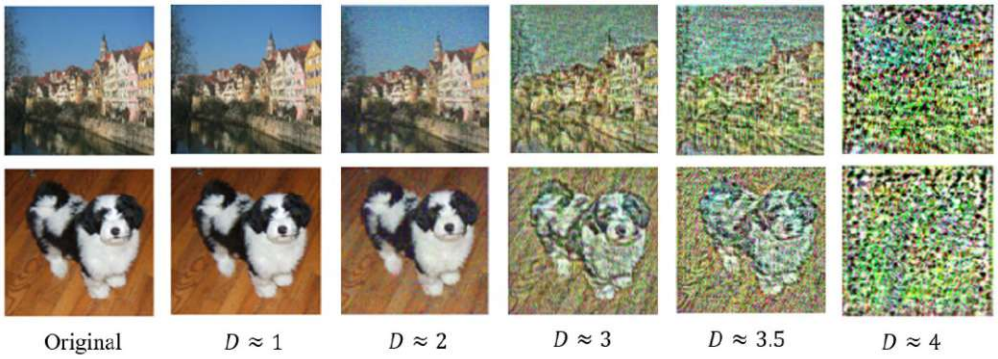

Figure 6.3: Quality of reconstruction for intermediate features from VGG19 Loss network, measured with image difference, adjusted from [OL19]

shows both images and their reconstruction results for each partition point. Table 5.8 shows all values of image difference for each image and reconstructed image. For example the image difference for reconstructed versions of image 9 and 10 were 3.02 and 3.04 respectively, for *pool_0*, which are also once more presented in Figure 6.4. Comparing the generated images of Figure 6.4 to Figure 6.3 shows, that while reconstructions from *pool_0* resulted in image differences of 3.02 and 3.04, their reconstructions are subjectively better than those of Figure 6.3 with an image difference of ≈ 3.

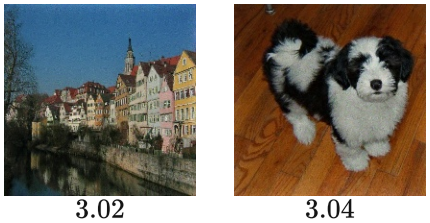

Figure 6.4: Generated images for partition point *pool_0* and images 9 and 10, with respective image difference value D

This may stem from the fact that another network was used in the works of [OL19]. This thesis focused on YoloV3-Tiny, while Figure 6.3 shows reconstruction results and image difference values for VGG19. As these networks are not using the same architecture, dimensions and values of intermediate features are also different. Therefore, as the image reconstruction attack relies on the intermediate features of the network, properties of reconstruction results, i.e., a green tint, may differ for both networks and is suspected to have caused the differences in generated images and resulting similarity metrics.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Suggested Thresholds

The average values of the three different metrics over all generated images were 4.1, 7.14, and 0.0077 for D_{Batch}, $mPSNR$, and $mSSIM$ for partition point *conv_3*. This partition point was the first to not expose any personally identifiable information (PII), as shown in Figures 5.17 and 5.18. Therefore, the lower bound of the values used to calculate the average can be used as a threshold. This results in 4.0, 8.0, and 0.001 for the respective similarity metric. Note that while for D the threshold is an upper limit, it is a lower limit for $PSNR$ and $SSIM$. In other words, values above 4.0 D, and values below 8.0 $PSNR$ and 0.001 $SSIM$ should not expose PII. If multiple partition points were subjected to the used reconstruction attack, the quality of resulting generated images could be automatically assessed with above similarity metrics. If the resulting metrics were exceeding the threshold, PII would be visible, which would help to find vulnerable partition points. This works fine in theory, but as the singular results of Tables 5.8, 5.9, and 5.10 show, outliers exist, which dampen the validity of the thresholds. For example, when considering the threshold of image difference D 4.0, the generated image for partition point *pool_2* resulted in an image difference D of 4.12 for *image 2*. Therefore, in this case, the image difference was exceeding the suggested threshold, yet PII were still visible on the generated image, as visible in Figure 5.17. These outliers can be found for both $SSIM$ and $PSNR$ as well. For *image 8* and *pool_2*, $SSIM$ had a value of 0.0097, while for *image 6* and *pool_2*, $PSNR$ had a value of 7.89. Both these values exceeded the suggested thresholds, but exposed PII as visible in Figure 5.18. These results show, that the thresholds should rather be used as a suggestion, as outliers exist.

Reconstructions for Max Pooling and Convolutional Layers

The generated images suggest, that reconstruction for partition points ending on a *max pooling* layer are nearly as good as those of the previous *convolutional* layer, when qualitatively evaluating if the generated image includes PII. This behavior can be explained, and seems pretty intuitive when considering that the attack uses intermediate features as its input. For this, the intermediate features of a *convolutional* layer have to be compared to those of the following *max pooling* layer. Max pooling layers, as explained in Section 3.5.2, typically reduce the size of their input by a factor of four. Additionally, the values of its input are not modified, which means that one quarter of the original intermediate features of the previous *convolutional* layer remain. Therefore, it makes sense, that the generated images are nearly the same for the *convolutional* and following *max pooling* layer, as a quarter of the original intermediate features are still available to be used for reconstruction.

In comparison to this behavior, reconstructions from partition points ending on *convolutional* layers, greatly differ from the reconstructions of the previous *max pooling* layers. This can be explained by the fact that *convolutional* layers actually modify their input.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

6.2.2 Progression of Image Difference

As explained in Section 3.5.2, 500 iterations of the attack are executed for each partition point. All executions start with a randomized image variable, which is why every reconstruction starts with the value of $\approx$ 4.25, 7, and 0 for D, $PSNR$ and $SSIM$. The progression in image similarity metrics for the different partition points, which was mentioned in Section 5.2.3, suggests that the reconstructed images could be improved even more for partition points *conv_0* and *pool_0* with more iterations. As these were already identified as partition points which allow for the reconstruction attack to expose personally identifiable information (PII), it is not really important that reconstructed images could be even more identical to the original image. It would have been interesting, if for any of the partition points where the reconstruction did not yield images exposing PII, room for improvement in reconstructions is possible. The stagnation after $\approx$ 200 iterations for partition points *conv_1* to *conv_4* suggest that additional iterations of the image reconstruction attack can not improve the reconstructed images for these partition points.

6.2.3 Time Requirements

The average reconstruction time of $\approx$ 5 seconds for all partition points shows that the reconstruction attack can be performed in an arguably short amount of time with a consumer-grade GPU. These results match those of Oh et al. [OL19], as their average was 7.7 seconds for 500 iterations. The results suggest, that an image exposing PII could be generated within a short time, if intermediate features from a partition point vulnerable to the image reconstruction attack were to be obtained. While this may hold true for a single frame of a video, generating a video based on an obtained transmission of a whole video stream, may take much longer, as the reconstruction attack would need 4-6 seconds for a single frame. Most of the time though, a single frame could be enough to expose PII, e.g., if a person was at a specific place at a specific time. The required time could be even lower, if less iterations were performed within the reconstruction attack, as the improvement of generated image stagnated after $\approx$ 200 iterations.

6.2.4 Requirements of an Image Reconstruction Attack

Lastly, as already explained in Section 3.5.2, the image reconstruction attack applied within this thesis comes with some requirements. First, the adversary would have to know which machine learning model is used within a Co-Inference VAP. Second, the adversary would have to gain access to a copy of the machine learning model. These two conditions can only be fulfilled under certain circumstances. If the model is a completely custom developed model for example, which is not available for public use, obtaining the model would be difficult. In the case of YoloV3-Tiny, the adversary could download it from online repositories. But that assumes, that the adversary is able to identify the used machine learning model. This would theoretically be possible, by comparing the shape of obtained intermediate features to those of layers in other machine learning models.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

If the output of a layer of another Convolutional Neural Network (CNN) matches the shape of the intercepted intermediate features, it could either identify the CNN which produced the intermediate features, or be a coincidence. The architectural structure of many machine learning models is available online, like Figure 3.5 showing the architecture of YoloV3-Tiny, which could be used for above comparison.

He et al. [HZL19] categorized attacks such as those of Oh et al. [OL19] as white box attacks, in which the adversary has to have access to the used machine learning model. Other attack types, which were mentioned by He et al. [HZL19], do not have as strong requirements as white box attacks. In black-box attacks, the adversary is only able to query the machine learning model, i.e., he/she is able to send an input to the model and intercept the intermediate features. In this case, the adversary first sends images to the machine learning model and intercepts the intermediate features. This is used to generate training data in form of *[intermediate features, original image]*. Then, the adversary can train an Inverse-Network with above training data. Dosovitskiy et al. [DB16] also perform such an Inverse-Network attack on AlexNet and compare the results to those of Autoencoders. Both black-box attacks yield reconstructions of varying quality, which are hard to compare to those of this thesis. This is due to both He et al. [HZL19] and Dosovitskiy et al. [DB16] choosing structurally simpler machine learning models, as LeNet-5, AlexNet and a custom CNN are evaluated. However, applying this Inverse-Network approach to YoloV3-Tiny is scheduled for future work and not in the scope of this thesis.

6.3 Summary of the Key Findings

In this section, the key findings are condensed and summarized. This summary is split into two subsections, each dealing separately with RQ_1 and RQ_2. Furthermore, the resulting trade-off between performance and data protection will be presented last.

6.3.1 Performance Benchmarks

To answer RQ_1, the different partition points of the Co-Inference VAP presented in Section 3.3 were evaluated based on throughput, resource utilization, and required bandwidth. The insights on how choice of partition point impacts these three performance aspects can help to choose partition points in future Co-Inference VAPs.

Throughput

Inference execution times of the Co-Inference VAP were collected to measure throughput. With these benchmarks, two trends in inference time were identified. The total inference time increased with partition point depth when the Raspberry Pi 3b+ was included in the Co-Inference VAP. For all other device-server pairs of the Co-Inference VAP, the total inference time decreased with partition point depth. As these results suggested that inference time was impacted by model type, all other available model types were evaluated

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

on the Xavier NX. These results further suggested, that above trends depend on the used model type. The partition points of YoloV3-Tiny alternately ended on *convolutional* and *max pooling* layers. This characteristic explained the alternating decrease and increase in inference times of consecutive partition points.

Resource Utilization

In terms of RAM usage, the results suggest that the choice of partition point has no impact on RAM usage on both devices partaking in Co-Inference. CPU usage on the other hand increased with partition point depth on the edge devices, which suggests that shallower partition points are more capable to alleviate resource constrained edge devices from computational overhead. Edge servers on the other hand showed two different trends in CPU usage. While it decreased for the Desktop PC, it increased for the ASRock 4x4. These results showed how accelerator availability can ease CPU usage, as the Desktop PC used its GPU to perform inference, while the ASRock 4x4 used its CPU.

Bandwidth Usage

The bandwidth analysis has shown that a significant reduction in bandwidth can be achieved by selecting the correct partition point. Partition points that end on a *max pooling* layer reduce the size of intermediate features by a factor of 4, when compared to the previous partition point. Therefore, partition points ending on a *max pooling* layer should be prioritized for Co-Inference VAPs which require low bandwidth requirements. The results further confirm that choice of partition point does not significantly impact the size of the final inference results, which are returned from the edge server.

Comparison of Local and Collaborative Inference VAP

All edge devices and servers partaking in the Co-Inference VAP were also benchmarked in a locally executed VAP. Comparing these benchmarks to that of the Co-Inference VAP showed that the Co-Inference VAP could outperform the Local Inference VAP in terms of total inference time. This was the case for 31 of the 72 different device-server pairs and partition points in the Co-Inference VAP, where total inference time was lower than that of the Local Inference VAP, as shown in Section 5.1.4. Comparing the resource utilization in the same manner suggests that Co-Inference results in a lower resource usage on both edge devices and servers in most cases, as shown in Section 5.1.4.

6.3.2 Data Protection

To answer RQ_2, the different partition points of the Co-Inference VAP from Section 3.3 were subjected to the image reconstruction attack presented in Section 3.5.2. For three of the nine partition points no images with personally identifiable information (PII) could be generated. For the other six partition points, images of varying quality could be generated, which subjectively exposed PII. It has to be noted that during literature research no method was found, that is able to objectively detect whether an image contains PII or

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

not, with 100% accuracy. Hence, an image generated for a specific partition point may still expose PII to some degree, which is subjective to the human observer. However, a Collaborative Inference approach still significantly reduces the chances of actually exposing PII when transmitting to and processing an image on another (potentially malicious) device, when the correct partition point is selected. These results suggest that Co-Inference can protect data when compared to traditional offloading, as the applied reconstruction attack failed to generate images exposing PII for three partition points.

The results of the image reconstruction attack were also measured objectively with three different metrics. With these metrics a possible threshold can be suggested for D, $PSNR$, and $SSIM$, which can help to validate if a partition point is prone to the image reconstruction attack. The average value of these metrics over all 10 images for a partition point were used to find a threshold. The averages were 4.1 for D_{Batch}, 7.14 for $mPSNR$, and 0.0077 for $mSSIM$ for partition point *conv_3* - the shallowest partition point exposing PII - which suggests thresholds of 4.0, 8.0, and 0.001 respectively.

The average time needed for reconstructing a single frame was 5 seconds on an Nvidia GTX 1060 6 GB. This suggests that if a partition point is vulnerable to the image reconstruction attack, reconstruction can be done in a short amount of time. The image similarity metrics were recorded for all 500 iterations that were performed in the image reconstruction attack. These progressions showed, that after $\approx$ 200 iterations, no improvements could be observed for the image similarity metrics, i.e., the generated image stopped improving in similarity. This suggests, that the image reconstruction attack could be stopped earlier, while still producing the same results.

6.3.3 Combined Findings

Above sections dealt with both research questions separately, although the results suggest that there is a trade-off between resource utilization, bandwidth requirements, data protection, and inference time. When choosing deeper partition points, both bandwidth requirements are reduced, and data is better protected against the applied reconstruction attack. For edge devices with accelerators the inference time is also lower with deeper partition points. When choosing shallower partition points though, edge devices without accelerators achieve a lower total inference time than with deeper partition points, while also reducing resource usage in general. These results suggest, that to ensure the best data protection, as mandated by the General Data Protection Regulation [Cou22], deeper partition points should be preferred.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 7

Conclusion

This chapter concludes the thesis by summarizing the contributions and key findings. First, the domain, motivation, and problem statement of the thesis are revisited. Second, the results of the experiments and their contributions are summarized for each research question. Lastly, an outlook on future work is presented.

7.1 Domain, Motivation, and Problem Statement

Edge Intelligence facilitates the execution of AI-based applications at the edge of the network, by combining principles of edge computing and Artificial Intelligence. A prominent example of such applications are Video Analysis Pipelines (VAPs), which utilize state-of-the-art machine learning models to generate specific insights on video data. Traffic monitoring is a typical use case of a VAP, where video streams are analyzed and, e.g., traffic accidents are automatically detected and further handled by an operating party. However, AI-based tasks, i.e., inference, needed for such advanced and autonomous video analysis are computationally expensive. The heterogeneous infrastructure at the edge often comprises devices that are limited in their computing capabilities. Those resource constraints pose various challenges to either functional (performance) or non-functional (data protection) aspects of a VAP. Offloading the processing of video streams to more powerful servers is a popular way to mitigate computational limitations. However, due to this offloading, receiving servers could use the raw video data with malicious intent, which imposes privacy concerns.

Collaborative Inference offers a novel way to mitigate both performance and privacy issues. The machine learning model used for inference is split up at a specific partition point, resulting in a so-called head and tail model. An edge device then performs inference with the head model and transmits only intermediate features to a server. The server then finishes inference using the tail model and returns proper inference results to the edge device. Partition point selection has a major impact on performance and data

protection. Deeper partition points can increase the workload on edge devices, while intermediate features of shallower partition points are more prone to sophisticated attacks, which aim to reconstruct original images from intermediate features. These attacks could possibly expose personally identifiable information of individuals visible on the original data, therefore violating privacy protection laws, such as the General Data Protection Regulation (GDPR). Hence, this thesis focuses on the impact different partition points have on the performance and privacy of Co-Inference.

7.2 Research Questions

This section states the key findings of this thesis and summarizes how the respective research questions were answered. Furthermore, the contributions and impact of these findings will be presented.

RQ$_1$: How do the different possible partition points of a Co-Inference VAP impact the performance in regard to throughput, resource utilization, and required bandwidth?

A VAP was implemented with Python and TensorFlow to provide the foundation needed to answer this question. YoloV3-Tiny was chosen as the machine learning model performing the task of object detection in this VAP, as it is geared towards resource constrained devices. To enable the evaluation of a Co-Inference execution of the VAP, YoloV3-Tiny was split at nine partition points into head and tail models. The testbed featured a representative array of devices, which reflects the heterogeneous nature of edge computing, as shown in Section 4.2. YoloV3-Tiny, and all head and tail models, were quantized and converted into TensorFlow Lite, Edge TPU, and TF-TRT models to be able to support the devices of the testbed optimally. The VAP was executed on each device of the testbed locally, i.e., alone, and benchmarked to provide a comparison baseline. With the head and tail models, and the devices of the testbed, the Co-Inference VAP and its nine partition points were benchmarked on eight different device-server pairs in terms of throughput, resource utilization, and required bandwidth.

The resulting benchmarks have shown that the choice of partition point has a major impact on all three performance aspects. Further, it has been shown in Section 6.1.2, that the type of last layer included in partition point, i.e., *convolutional* or *max pooling* layer, and the type of model used for inference, e.g., TensorFlow, TensorFlow Lite, also impacts these performance aspects. Throughput of the Co-Inference VAP with accelerator-equipped edge devices increased with partition point depth, while it decreased for those with the Raspberry Pi 3B+. Section 6.1.3 showed that the resource utilization in terms of RAM usage was approximately the same for all partition points on edge devices and servers. It further showed that CPU usage increased with partition point depth for edge devices, and edge servers performing inference on the CPU, while it decreased for accelerator-equipped edge servers. Bandwidth requirements decreased with partition point depth, resulting in a 32× reduction when comparing the highest and lowest bandwidth requirements of the different partition points, as shown in Section 5.1.3. With increasing depth of layers in a Convolutional Neural Network, such as YoloV3-Tiny, their

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

respective output decreases in terms of size. Hence, deeper partition points will generally have lower bandwidth requirements than shallower ones. Some device-server pairs were also able to achieve higher throughput in the Co-Inference VAP, when compared to the edge device performing the VAP alone locally, as shown by the results in Section 5.1.4.

These performance benchmarks can be used to select an adequate partition point in Co-Inference VAPs at design time, which want to utilize YoloV3-Tiny to facilitate their analysis. Depending on the requirements and system composition of a Co-Inference VAP, an optimal partition point can be selected based on the results of this thesis. For example, the lowest possible bandwidth requirements can be achieved with partition point *pool_3*.

RQ$_2$: How do the different possible partition points of a Co-Inference VAP impact security and data protection?

Image reconstruction attacks are sophisticated methods, which aim to reconstruct original images from their intermediate features. As YoloV3-Tiny was split at nine partition points, the intermediate features generated by each of the resulting head models differed from each other when performing inference on the same input. Therefore, intermediate features were generated with each of the head models for ten images. These intermediate features were then used as the input for an image reconstruction attack, generating 90 images in total. This was performed to evaluate if generated images would expose personally identifiable information (PII) visible on the original data, i.e., if transmission of intermediate features could provide data protection in a Co-Inference VAP when selecting a specific partition point. To objectively evaluate the similarity of generated images to the original images, three different state-of-the-art metrics were used, which were presented in Section 3.5.3.

The results of the image reconstruction attack heavily depended on the chosen partition point, as shown by the results in Section 5.2. For six partition points the image reconstruction attack generated images exposing PII of the original images. For the remaining three partition points the image reconstruction attack generated subjectively unrecognizable images. Thresholds for the image similarity metrics were presented in Section 6.2.1 based on the average values of the image similarity metrics, which can be used as a guideline, to automatically verify if a generated image does expose PII.

As the generated images for three partition points did not expose PII, they did in fact protect the exchanged data against the applied reconstruction attack. In other words, the server (or an adversary) receiving these intermediate features would not be able to reconstruct the original video data with the applied reconstruction attack for these three partition points. In comparison to typical offloading, where the server is able to view the received data in full, Co-Inference would therefore protect the data against malicious intent. Concerns regarding privacy are further exacerbated by the GDPR, which requires that *"personal data shall be processed in a manner that ensures appropriate security of the personal data, including protection against unauthorised or unlawful processing"* [Cou22, Ch. II, Art. 5, 1.(f)]. The results of this thesis have shown that deeper partition points are more capable of complying with this regulation. As this thesis used YoloV3-Tiny

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

in the VAP, image reconstruction results for the partition points are naturally only applicable for this machine learning model. Other models would have to be subjected to this image reconstruction attack as well, to evaluate their resistance to this attack.

This thesis analyzed both the performance and privacy aspects of Co-Inference together and conducted the evaluation based on multiple partition points. Therefore, this thesis aimed to close the gap of existing research between performance and privacy in Co-Inference. The results showed that deeper partition points reduce bandwidth requirements and are able to withstand the applied reconstruction attack. Shallower partition points on the other hand achieve a lower total inference time on edge devices without accelerators, while also reducing resource usage in general.

7.3 Future Work

This section presents possible future work that would alleviate limitations of this thesis. During conduction of the experiments, several theories and ideas have been brought up that require further research, which were presented in the discussion in Chapter 6.

7.3.1 Performance Benchmarks with Simulated Latency

As described in Sections 4.3 and 6.1.1, latency was not dealt with in this thesis, as it is heavily affected by environmental factors, such as signal strength, signal quality, interferences, and many more. This latency could be included in future research as well, by e.g., simulating the latency, to allow for more insights on the performance of Co-Inference VAPs. With these simulations, possible thresholds of latency could be found for the execution of Co-Inference. These thresholds would help to predict whether a Co-Inference VAP can provide better performance than executing the analysis on an edge device alone, based on present latency.

7.3.2 Low Level Profiling

In Sections 6.1.2 and 6.1.2, it was suggested that size of intermediate features could impact the inference time. This was further suggested to be caused by memory operations, necessary to transfer intermediate features from accelerator or machine learning model memory to main memory after finishing inference. To confirm this suspicion low level profiling of memory operations would be required.

7.3.3 Additional Image Reconstruction Attacks

This thesis applied the image reconstruction attack presented by Oh et al. [OL19]. Other reconstruction attacks could be applied to the partition points of the Co-Inference VAP, to further validate the results of this thesis. One candidate would be the Inverse-Network technique suggested by He et al. [HZL19], which trains a model to facilitate the attack. Autoencoders could also prove to generate better results, which were suggested and evaluated by Dosovitskiy et al. [DB16], and should therefore also be implemented.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

APPENDIX A

Remaining Benchmarks

A.1 Co-Inference VAP Performance Benchmarks

This section includes all the remaining performance benchmarks of the Co-Inference VAP which were not included in Section 5.1.2. Section A.1.1 shows the inference times while Section A.1.2 shows the resource utilization of various device-server pairs executing the Co-Inference VAP.

A.1.1 Inference times

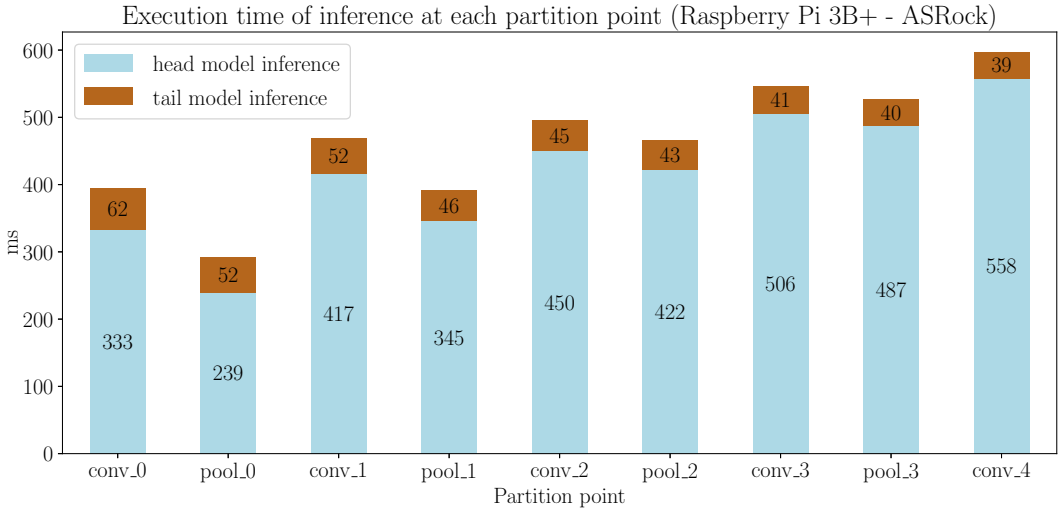

Figure A.1: Co-Inference execution time on the Raspberry Pi 3B+ and the ASRock 4x4

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

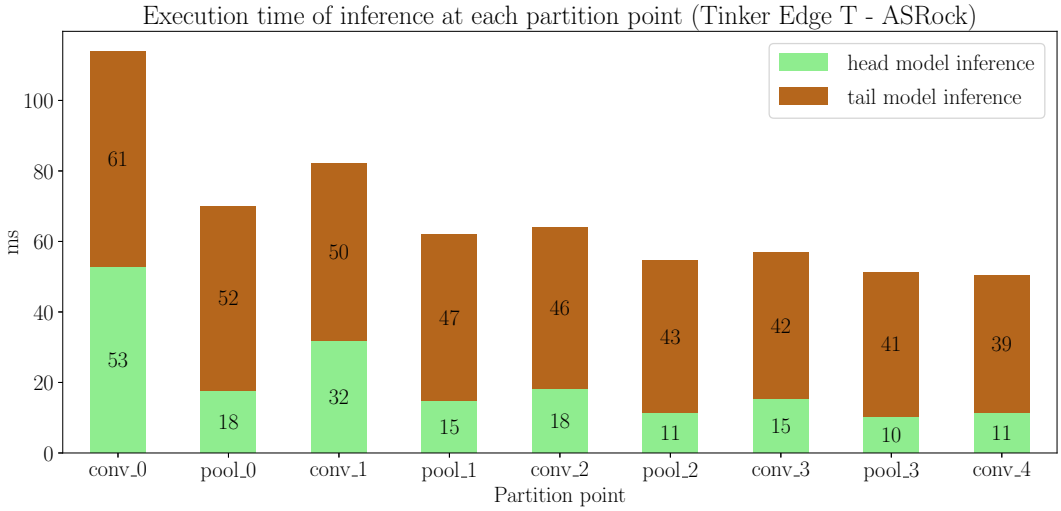

Figure A.2: Co-Inference execution time on the Tinker Edge T and the ASRock 4x4

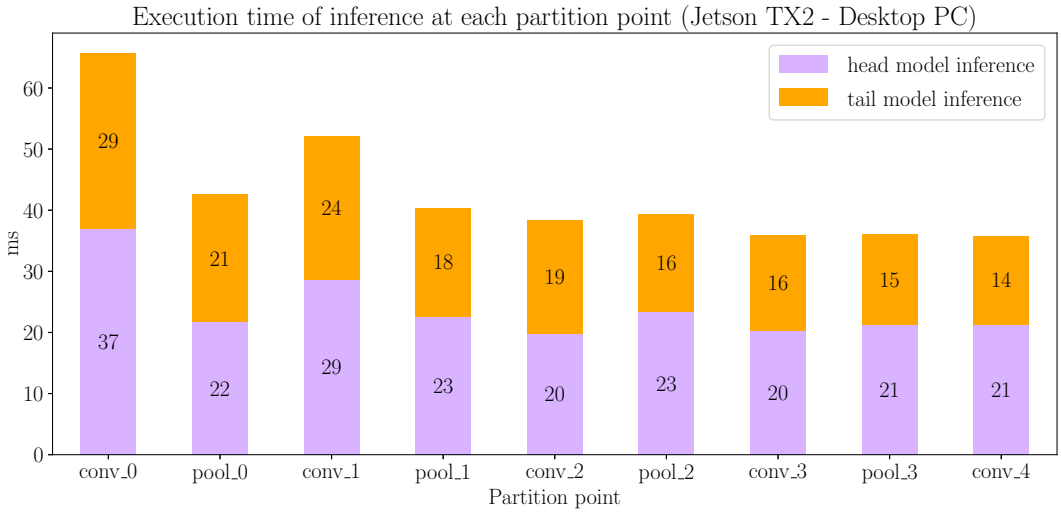

Figure A.3: Co-Inference execution time on the Jetson TX2 and the Desktop PC

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

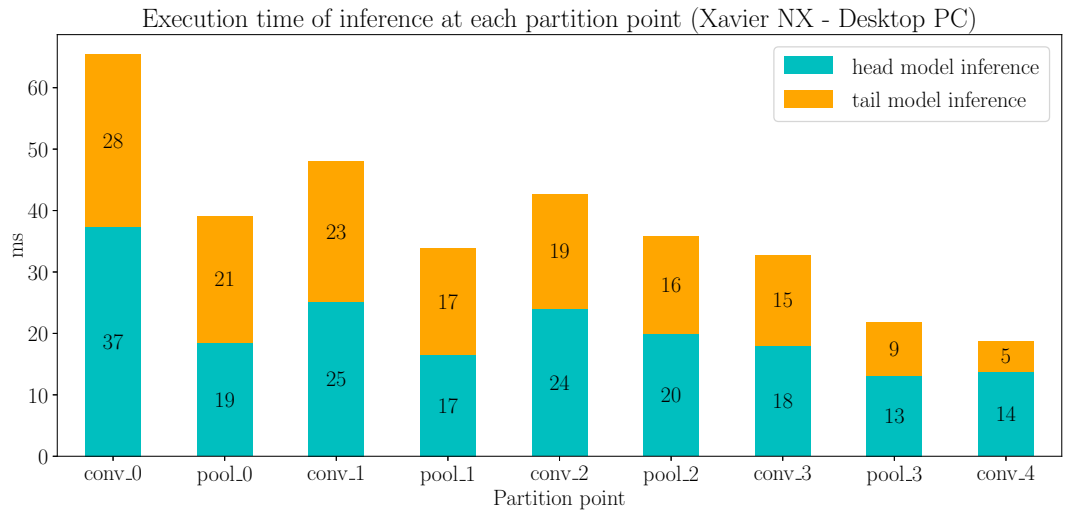

Figure A.4: Co-Inference execution time on the Xavier NX and the Desktop PC

A.1.2 Resource Utilization

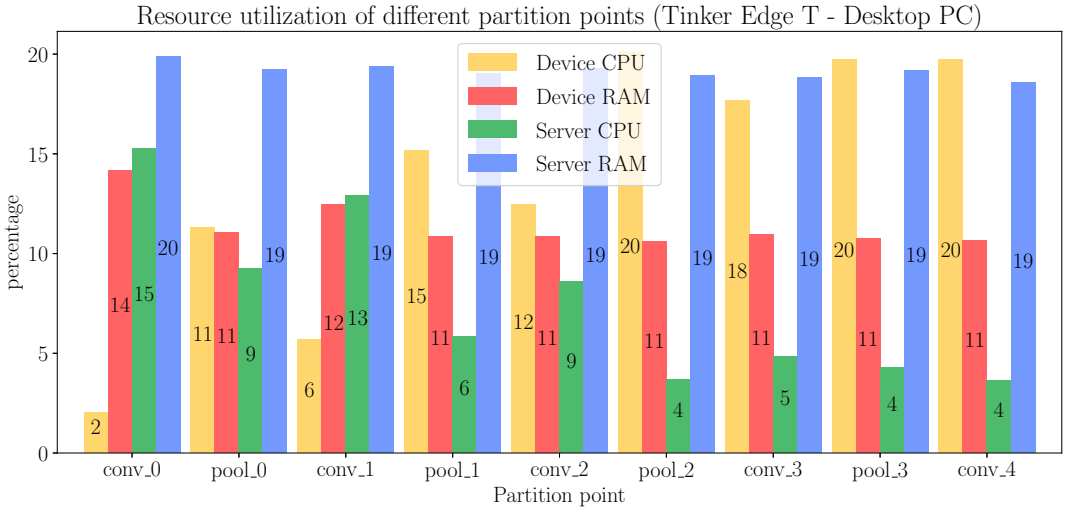

Figure A.5: Resource utilization per partition point on the Tinker Edge T and the Desktop PC

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

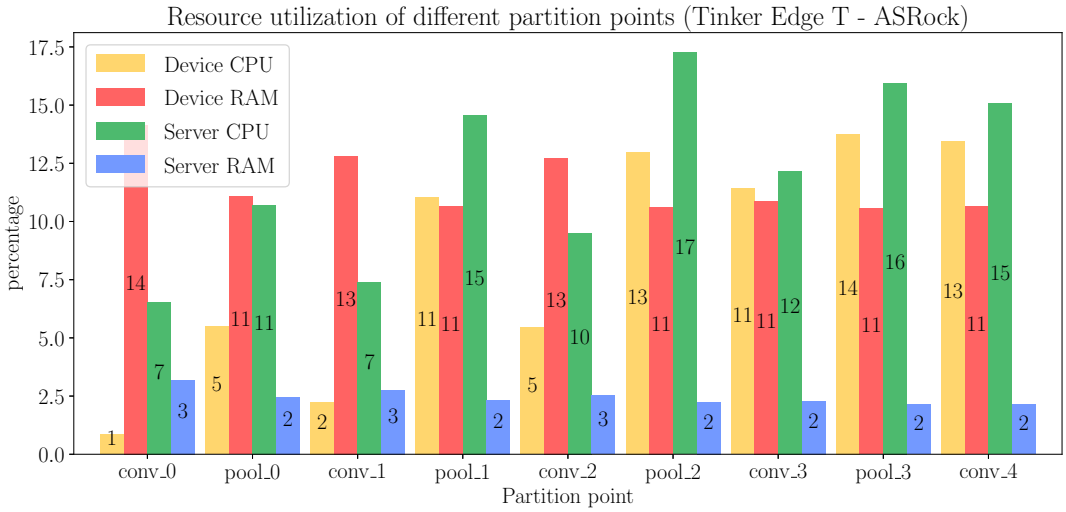

Figure A.6: Resource utilization per partition point on the Tinker Edge T and the ASRock 4x4

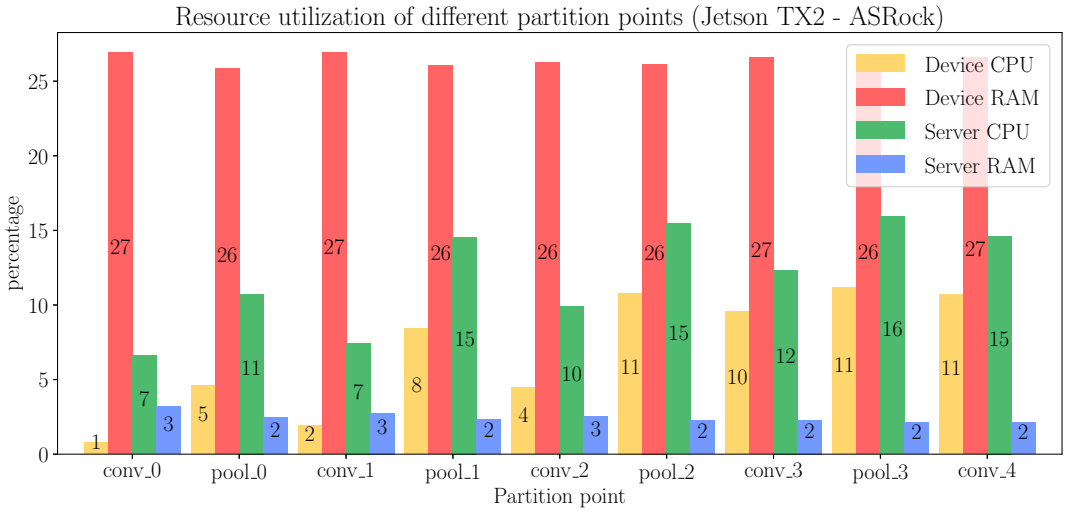

Figure A.7: Resource utilization per partition point on the Jetson TX2 and the ASRock 4x4

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

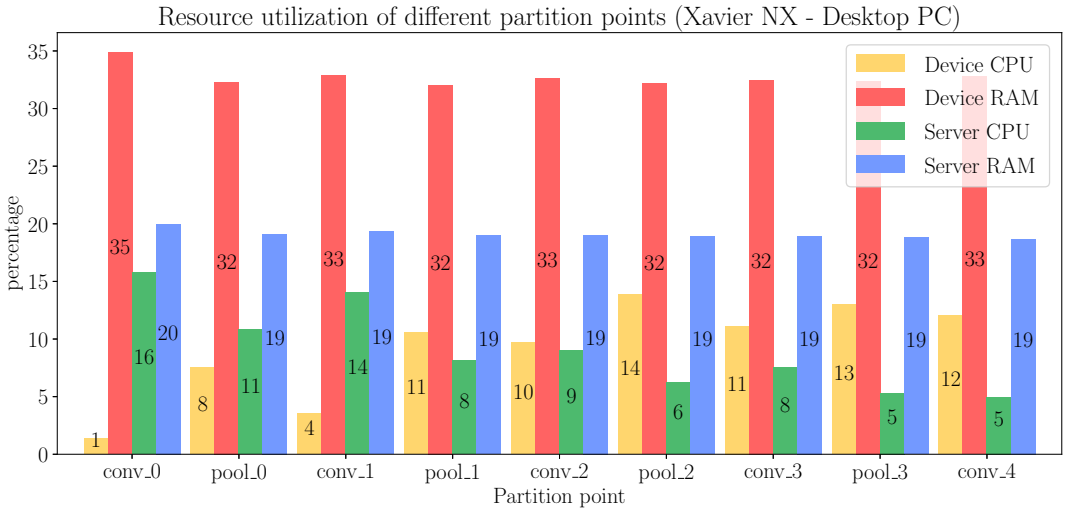

Figure A.8: Resource utilization per partition point on the Xavier NX and the Desktop PC

A.2 Comparison of Local and Collaborative Benchmarks

This section shows Table A.1, comparing the mean inference time of all devices executing the Local Inference VAP (local row) and all device-server pairs executing the Co-Inference VAP. The cell is marked as green if the inference time was lower in the Co-Inference VAP.

Edge Device **Edge Server**	RBP DPC	RBP ASR	TET DPC	TET ASR	TX2 DPC	TX2 ASR	XNX DPC	XNX ASR
local	1519 ms	1519 ms	56 ms	56 ms	36 ms	36 ms	30 ms	30 ms
conv_0	357 ms	395 ms	81 ms	114 ms	66 ms	103 ms	65 ms	101 ms
pool_0	262 ms	292 ms	39 ms	70 ms	43 ms	74 ms	39 ms	72 ms
conv_1	423 ms	469 ms	55 ms	82 ms	52 ms	84 ms	48 ms	77 ms
pool_1	358 ms	391 ms	31 ms	62 ms	40 ms	70 ms	34 ms	64 ms
conv_2	469 ms	495 ms	34 ms	64 ms	38 ms	71 ms	43 ms	69 ms
pool_2	439 ms	465 ms	16 ms	55 ms	39 ms	68 ms	36 ms	64 ms
conv_3	513 ms	546 ms	19 ms	57 ms	36 ms	68 ms	33 ms	58 ms
pool_3	503 ms	527 ms	22 ms	51 ms	36 ms	60 ms	22 ms	55 ms
conv_4	578 ms	597 ms	16 ms	50 ms	36 ms	60 ms	19 ms	55 ms

Table A.1: Mean inference time for each edge device in the Local Inference VAP, and mean total (head + tail) inference time for the device-server pairs in the Co-Inference VAP

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

A.3 Resource Utilization of Model Types

This section presents the resource utilization benchmarks of various model types on the Xavier NX, of which the inference benchmarks were already presented in Section 5.1.2.

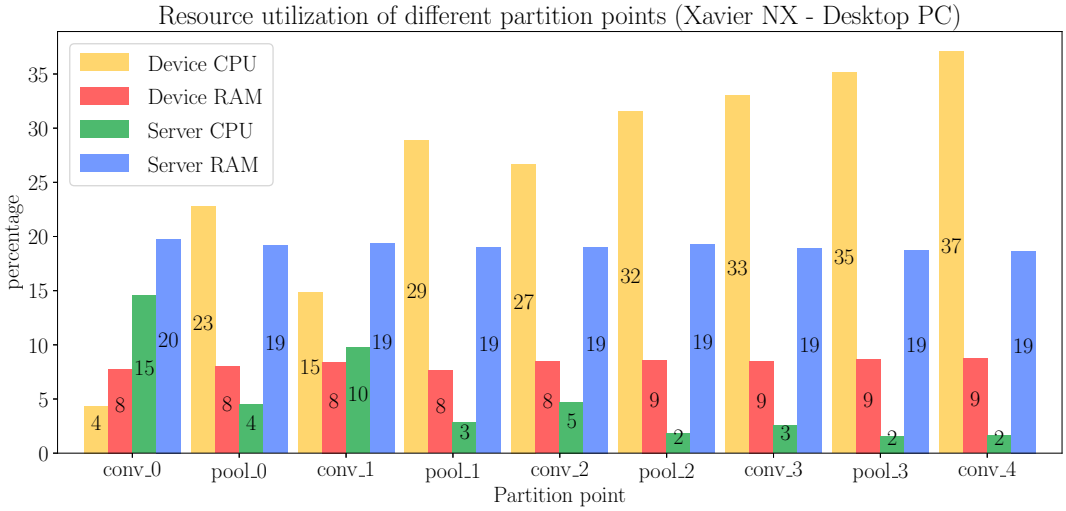

Figure A.9: Resource utilization per partition point on the Xavier NX and the Desktop PC for TensorFlow model execution on CPU

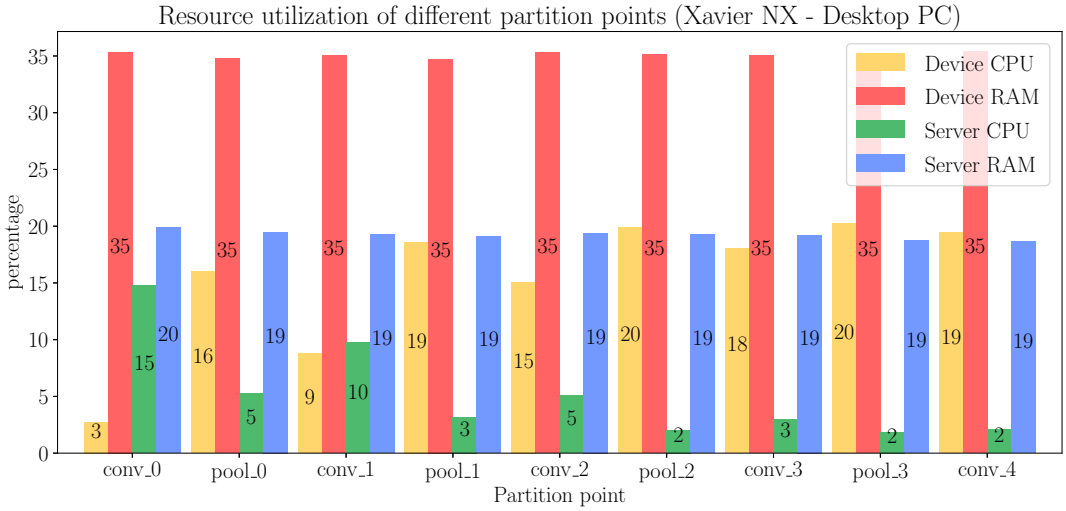

Figure A.10: Resource utilization per partition point on the Xavier NX and the Desktop PC for TensorFlow model execution on GPU

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

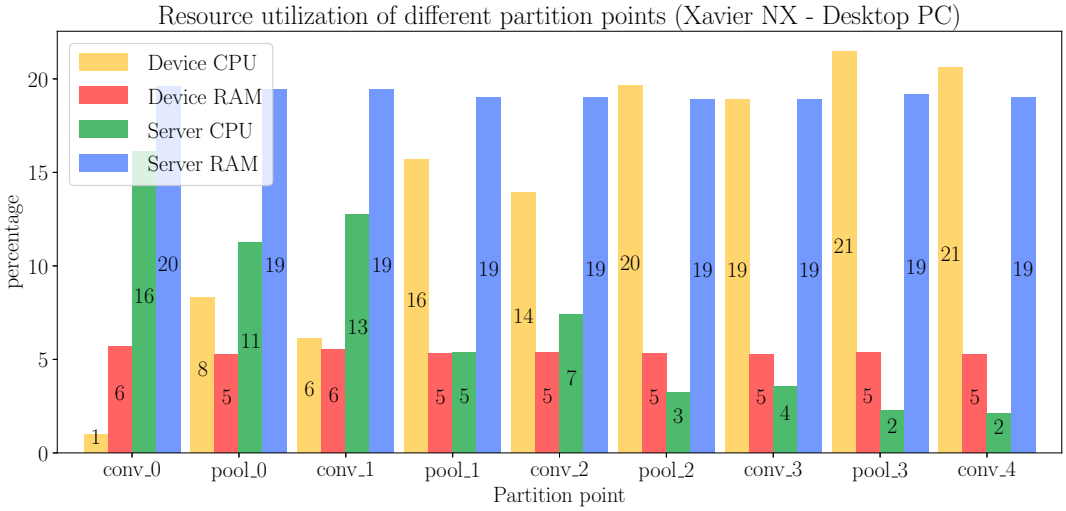

Figure A.11: Resource utilization per partition point on the Xavier NX and the Desktop PC for TensorFlow Lite model execution

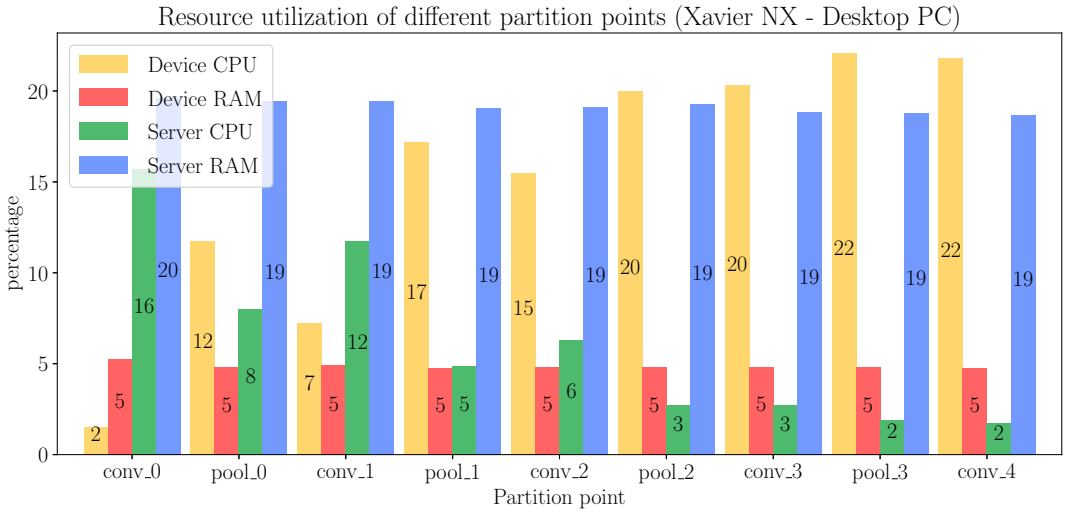

Figure A.12: Resource utilization per partition point on the Xavier NX and the Desktop PC for TensorFlow Lite 8-bit quantized execution

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

2.1 Edge computing overview, Shi and Dustdar [SD16] - Edge devices can generate and consume data, with nodes along the edge taking on some computational workload, e.g., data caching, data processing, with the end goal of storing data in the cloud 6
2.2 Neuron, adapted from [Nie15] 8
2.3 Fully connected neural network 9
2.4 2D Kernel with size 3, stride length 1, sliding across a 5x5 image, adapted from [Nie15] 10
2.5 Pooling with filter size 2, stride length 2, sliding across a 4x4 matrix, adapted from [Nie15] 10
2.6 Architecture of YOLO, a CNN for object detection [RDGF16] 11
2.7 Conversion pipeline to generate a model for inference on an Edge TPU . . 13
2.8 Offloading Categories, adapted from [KHG+17] 14
2.9 CNN with route layer and partition point in red 15
2.10 AlexNet layer inference and transmission cost analysis, by Li et al. [LZC18] 15

3.1 A single frame of the implemented VAP with YoloV3-Tiny - sample video from Shutterstock [JL22] 21
3.2 Process of the Local Inference VAP for each frame 22
3.3 Process of the Co-Inference VAP for each frame 22
3.4 YoloV3-Tiny architecture, adjusted from [FWR19] 23
3.5 YoloV3-Tiny network structure table, from [HHW+19] 24
3.6 Calculation of image reconstruction loss of i-th layer, from [OL19]. 27

4.1 Images used in the reconstruction attack - Image 1 to 8 are from the Pascal VOC 2007 test data set, image 9 and 10 were also used in the paper of Oh et al. [OL19] 34

5.1 Mean local inference time on test devices per frame 36
5.2 Mean resource utilization of the test devices during the Local Inference VAP 37
5.3 Co-Inference execution time on the Raspberry Pi 3B+ and the Desktop PC 38
5.4 Co-Inference execution time on the Tinker Edge T and the Desktop PC . 39
5.5 Co-Inference execution time on the Jetson TX2 and the ASRock 4x4 . . . 39
5.6 Co-Inference execution time on the Xavier NX and the ASRock 4x4 . . . 40

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

5.7 Co-Inference execution time on CPU per partition point on the Xavier NX 41
5.8 Co-Inference execution time on GPU per partition point on the Xavier NX 41
5.9 Co-Inference execution time per partition point on the Xavier NX with TensorFlow Lite . . . 42
5.10 Co-Inference execution time per partition point on the Xavier NX with TensorFlow Lite 8-bit quantized . . . 42
5.11 Resource utilization per partition point on the Raspberry Pi 3B+ and the Desktop PC . . . 43
5.12 Resource utilization per partition point on the Jetson TX2 and the Desktop PC . . . 44
5.13 Resource utilization per partition point on the Xavier NX and the ASRock 4x4 . . . 45
5.14 Resource utilization per partition point on the Raspberry Pi 3B+ and the ASRock 4x4 . . . 45
5.15 Size of intermediate features per partition point . . . 46
5.16 Mean size of inference results per partition point . . . 46
5.17 Reconstructions for images 1-5 for each partition point, including D_{Batch} over all 10 images . . . 51
5.18 Reconstructions for images 6-10 for each partition point, including D_{Batch} over all 10 images . . . 52
5.19 D_{Batch} for each partition point . . . 53
5.20 $mPSNR$ for each partition point . . . 54
5.21 $mSSIM$ for each partition point . . . 54
5.22 Average time needed for reconstructing from each partition point with the GTX1060 6 GB of Desktop PC . . . 55
5.23 D_{Batch} after every iteration for each partition point . . . 56
5.24 $mPSNR$ after every iteration for each partition point . . . 56
5.25 $mSSIM$ after every iteration for each partition point . . . 57

6.1 Latency of AlexNet partition points, by Kang et al. [KHG+17] . . . 61
6.2 Per layer execution time and size of intermediate features for VGG, by Kang et al. [KHG+17] . . . 64
6.3 Quality of reconstruction for intermediate features from VGG19 Loss network, measured with image difference, adjusted from [OL19] . . . 69
6.4 Generated images for partition point *pool_0* and images 9 and 10, with respective image difference value D . . . 69

A.1 Co-Inference execution time on the Raspberry Pi 3B+ and the ASRock 4x4 79
A.2 Co-Inference execution time on the Tinker Edge T and the ASRock 4x4 . 80
A.3 Co-Inference execution time on the Jetson TX2 and the Desktop PC . . . 80
A.4 Co-Inference execution time on the Xavier NX and the Desktop PC . . . 81
A.5 Resource utilization per partition point on the Tinker Edge T and the Desktop PC . . . 81

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

A.6 Resource utilization per partition point on the Tinker Edge T and the ASRock 4x4 82
A.7 Resource utilization per partition point on the Jetson TX2 and the ASRock 4x4 82
A.8 Resource utilization per partition point on the Xavier NX and the Desktop PC 83
A.9 Resource utilization per partition point on the Xavier NX and the Desktop PC for TensorFlow model execution on CPU 84
A.10 Resource utilization per partition point on the Xavier NX and the Desktop PC for TensorFlow model execution on GPU 84
A.11 Resource utilization per partition point on the Xavier NX and the Desktop PC for TensorFlow Lite model execution 85
A.12 Resource utilization per partition point on the Xavier NX and the Desktop PC for TensorFlow Lite 8-bit quantized execution 85

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Tables

3.1 Properties of the sample video used for evaluation of the VAP 20
3.2 Selected partition points for YoloV3-Tiny 24
3.3 Available models for evaluation 25

4.1 Testbed devices 30
4.2 Timestamps and respective steps of the Co-Inference VAP 31
4.3 Co-Inference VAP benchmark calculation 31
4.4 Test devices and machine learning model pairing 32
4.5 Timestamps and respective steps of the Local Inference VAP 32
4.6 Local Inference VAP benchmark calculation 32

5.1 Average time needed for resizing an image in the Local Inference VAP for each device of the testbed 36
5.2 Mean inference time for each edge device in the Local Inference VAP as the baseline (100%), and relative mean total (head + tail) inference time for the device-server pairs in the Co-Inference VAP 47
5.3 CPU usage of edge devices in the Local Inference VAP and in the Co-Inference VAP for each partition point 48
5.4 RAM usage of edge devices in the Local Inference VAP and Co-Inference VAP for each partition point 48
5.5 CPU usage of edge servers in the Local Inference VAP and Co-Inference VAP for each partition point 49
5.6 RAM usage of edge servers in the Local Inference VAP and Co-Inference VAP for each partition point 49
5.7 Average image quality metrics for all images 53
5.8 Image difference D for all generated images for all partition points (PP) . 58
5.9 $PSNR$ for all generated images for all partition points (PP) 58
5.10 $SSIM$ for all generated images for all partition points (PP) 58

A.1 Mean inference time for each edge device in the Local Inference VAP, and mean total (head + tail) inference time for the device-server pairs in the Co-Inference VAP 83

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

[ABB+17] Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. Real-time video analytics: The killer app for edge computing. *computer*, 50(10):58–67, 2017.

[Cou22] Council of European Union. Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation). `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:02016R0679-20160504`, 2016 (accessed Jan 3, 2022).

[DB16] Alexey Dosovitskiy and Thomas Brox. Inverting visual representations with convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4829–4837, 2016.

[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 of Things Journal*, 7(8):7457–7469, 2020.

[FWR19] Wei Fang, Lin Wang, and Peiming Ren. Tinier-yolo: A real-time object detection method for constrained environments. *IEEE Access*, 8:1935–1944, 2019.

[HHW+19] Wangpeng He, Zhe Huang, Zhifei Wei, Cheng Li, and Baolong Guo. Tf-yolo: An improved incremental network for real-time object detection. *Applied Sciences*, 9(16):3225, 2019.

[HZ10] Alain Hore and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *2010 20th international conference on pattern recognition*, pages 2366–2369. IEEE, 2010.

[HZL19] Zecheng He, Tianwei Zhang, and Ruby B Lee. Model inversion attacks against collaborative inference. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 148–162, 2019.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[Int21] International Data Corporation. Iot growth demands rethink of long-term storage strategies. https://www.idc.com/getdoc.jsp?containerId=prAP46737220, 2020 (accessed Dec 18, 2021).

[JL22] Shutterstock Jacob Lund. Office going pedestrians using their phones while commuting. commuters being on their mobile phones walking to office. https://www.shutterstock.com/video/clip-1022040100-office-going-pedestrians-using-their-phones-while, (accessed Jan 5, 2022).

[KHG+17] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News*, 45(1):615–629, 2017.

[LZC18] En Li, Zhi Zhou, and Xu Chen. Edge intelligence: On-demand deep learning model co-inference with device-edge synergy. In *Proceedings of the 2018 Workshop on Mobile Edge Communications*, pages 31–36, 2018.

[MB17] Pavel Mach and Zdenek Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials*, 19(3):1628–1656, 2017.

[MP43] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[MV15] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5188–5196, 2015.

[Nie15] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA, 2015.

[OL19] Hyunseok Oh and Youngki Lee. Exploring image reconstruction attack in deep learning computation offloading. In *The 3rd International Workshop on Deep Learning for Mobile Systems and Applications*, pages 19–24, 2019.

[RD19] Thomas Rausch and Schahram Dustdar. Edge intelligence: The convergence of humans, things, and ai. In *2019 IEEE International Conference on Cloud Engineering (IC2E)*, pages 86–96. IEEE, 2019.

[RDGF16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[RF18] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[RF22] Joseph Redmon and Ali Farhadi. Yolo: Real-time object detection. https://pjreddie.com/darknet/yolo/, 2018 (accessed Jan 10, 2022).

[RN02] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach. 2002.

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

[SD16] Weisong Shi and Schahram Dustdar. The promise of edge computing. *Computer*, 49(5):78–81, 2016.

[SPH+05] Andrew Senior, Sharath Pankanti, Arun Hampapur, Lisa Brown, Ying-Li Tian, Ahmet Ekin, Jonathan Connell, Chiao Fe Shu, and Max Lu. Enabling video privacy through computer vision. *IEEE Security & Privacy*, 3(3):50–57, 2005.

[SPX19] Weisong Shi, George Pallis, and Zhiwei Xu. Edge computing [scanning the issue]. *Proceedings of the IEEE*, 107(8):1474–1481, 2019.

[The22] The PASCAL Visual Object Classes Homepage. Pascal voc2007 challenge dataset. http://host.robots.ox.ac.uk/pascal/VOC/voc2007/, 2007 (accessed Jan 5, 2022).

[TMK17] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Distributed deep neural networks over the cloud, the edge and end devices. In *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*, pages 328–339. IEEE, 2017.

[TUR50] A. M. TURING. I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX(236):433–460, oct 1950.

[VF13] Ovidiu Vermesan and Peter Friess. *Internet of things: converging technologies for smart environments and integrated ecosystems*. River publishers, 2013.

[WBSS04] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[ZCL+19] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 107(8):1738–1762, 2019.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN