Ressourcenbasierte Inferenz-Partitionierung von Deep Neural Networks in eingebetteten Systemen

Zur Erlangung des akademischen Grades eines

DOKTORS DER INGENIEURWISSENSCHAFTEN (Dr.-Ing.)

von der KIT-Fakultät für Elektrotechnik und Informationstechnik
des Karlsruher Instituts für Technologie (KIT)
angenommene

DISSERTATION

von

M.Sc. Fabian Frederic Kreß
geboren am 12.07.1995 in Karlsruhe

Tag der mündlichen Prüfung: 19.12.2024

Hauptreferent: Prof. Dr.-Ing. Dr. h. c. Jürgen Becker
Korreferent: Prof. Dr.-Ing. Mladen Berekovic

Kurzfassung

In vielen Anwendungsbereichen, wie der Robotik oder im Automobilbereich, in denen eine große Anzahl von Sensoren integriert ist, sind eingebettete Systeme zu finden. Diese erzeugen oft große Datenmengen, die in möglichst kurzer Zeit von den sensornahen Plattformen an die zentrale Steuereinheit übertragen werden müssen. Folglich benötigen solche Plattformen hohe Linkbandbreiten für die Verbindungen zwischen den eingesetzten Komponenten. Der zunehmende Einsatz von Künstlicher Intelligenz (KI) in vielen dieser Anwendungen stellt darüber hinaus weitere Anforderungen an das Gesamtsystem. So erfordern insbesondere sicherheitskritische Anwendungen, wie das autonome Fahren, zur Verarbeitung von rechenintensiven Deep Neural Networks (DNNs) einen hohen Datendurchsatz bei gleichzeitig geringer Latenz. Aus diesem Grund wurden im Laufe des vergangenen Jahrzehnts verschiedenste analoge Processing-In-Memory-Architekturen (PIM) sowie rein digitale Hardwarebeschleuniger entwickelt, welche auf den Datenfluss von DNNs optimiert sind und damit eine bessere Performance und Energieeffizienz bieten als handelsübliche CPUs und GPUs. Modernste DNN-Architekturen bestehen jedoch mitunter aus mehreren Millionen Parametern, die für eine energie- und zeiteffiziente Inferenz möglichst nahe am Prozessor oder dem dedizierten Beschleuniger gespeichert werden müssen. Die verfügbare Fläche, insbesondere von sensornahen Mikroprozessoren, ist allerdings in der Regel stark begrenzt, sodass die internen Speicher dieser Plattformen den Anforderungen ganzer DNNs oft nicht vollständig genügen.

Ein Ansatz zur Lösung dieser Herausforderungen in eingebetteten Systemen mit mehreren zum Teil verteilten Hardwarebeschleunigern ist die Inferenz-Partitionierung von DNNs. Dabei wird die Arbeitslast über verschiedene Beschleuniger im System verteilt, um eine höhere Performance sowie Energieeffizienz zu erzielen und gleichzeitig die benötigte Fläche für die Hardwareressourcen über mehrere Plattformen aufzuteilen. Die Partitionierung von DNNs in eingebetteten Systemen ist allerdings eine komplexe Aufgabe, da hierbei verschiedene Optimierungsziele, wie die benötigte Bandbreite zur Übertragung von Zwischenergebnissen, beachtet werden müssen. Der aktuelle Stand der Technik umfasst unterschiedliche Ansätze, welche allerdings relevante

Metriken vernachlässigen oder lediglich auf bestimmte Rechnerarchitekturen zugeschnitten sind.

Die vorliegende Arbeit präsentiert daher mehrere neuartige Methoden zur Optimierung solcher Systeme auf unterschiedlichen Entwurfsebenen. Dazu zählt unter anderem ein generischer Ansatz auf Systemebene für die Inferenz-Partitionierung von DNNs mit dem Fokus auf Convolutional Neural Networks (CNNs), welcher eine Vielzahl an nicht-funktionalen Metriken sowie die Genauigkeit des CNNs als funktionale Metrik für die Optimierung verwendet. Letzteres ist notwendig, da Hardwarebeschleuniger in der Regel keine Fließkommazahlen, sondern meist Festkomma- oder gar Ganzzahlen verarbeiten. Die Quantisierung von Gewichten und Aktivierungen führt dabei üblicherweise zu einer Verschlechterung der Genauigkeit, weshalb auch diese Metrik bei der Suche nach einem optimierten Partitionierungsschema berücksichtigt werden muss. Das Framework CNNParted, das diese Methodik implementiert, exploriert automatisiert den Entwurfsraum der Inferenz-Partitionierung in Systemen mit mehreren Beschleunigern unter Beachtung der verfügbaren Hardwareressourcen. Dabei verwendet es Modellierungen der integrierten analogen PIM- und digitalen Hardwarebeschleuniger sowie der Links zwischen den Plattformen im System, um valide Partitionierungsschemata zu bewerten. Aus den Ergebnissen der Untersuchungen generiert CNNParted schließlich eine Pareto-Front hinsichtlich der Optimierungsziele. Die durchgeführten Experimente belegen, dass eine ressourcenbasierte Inferenz-Partitionierung vorteilhaft gegenüber Methoden des aktuellen Stands der Technik ist. So wurden zum Teil deutliche Verbesserungen verschiedener Metriken für heterogene Systeme, bestehend aus analogen PIM- und digitalen Hardwarebeschleunigern, erzielt. Folglich erreicht die entsprechende Wahl eines Pareto-optimalen Partitionierungsschemas unter anderem eine deutlich höhere Energieeffizienz als für die Ausführung der Inferenz auf einer einzigen Beschleunigerplattform.

Anschließend legt die vorliegende Arbeit aber auch dar, dass eine Inferenz-Partitionierung nicht in jedem Fall ohne zusätzliche Optimierungen sinnvoll ist. So ist die Einhaltung von Systemanforderungen meist nicht gewährleistet, wenn zur Übertragung von Zwischenergebnissen Links mit geringer Bandbreite zum Einsatz kommen. Folglich ist eine Optimierung der Software notwendig, um dennoch eine Inferenz-Partitionierung zu ermöglichen. Gleichzeitig muss aufgrund des begrenzten Energiebudgets in Anwendungsfällen wie dem Internet-of-Things (IoT) jedoch auch die Hardware optimiert werden, um DNNs zu integrieren. Handelsübliche Mikrocontroller verfügen meist nicht über die notwendige Performance und Energieeffizienz für die Ausführung der Inferenz. Da nach aktuellem Stand der Technik nur wenige Metho-

den existieren, welche ein entsprechendes Hardware/Software Co-Design durchführen, zeigt die vorliegende Arbeit weitere Ansätze zur Lösung dieser Probleme auf. So kann unter anderem durch Implementierung von approximierten Multiplizierern und einer approximierten Aktivierungsfunktion ein sehr kleiner Hardwarebeschleuniger realisiert werden, dessen Architektur bei Anwendung von Quantization-Aware Training (QAT) nur zu einem geringen Verlust der Genauigkeit des DNNs führt. Des Weiteren stellt die vorliegende Arbeit die Beschleunigerarchitektur LOTTA für Temporal Convolutional Networks (TCNs) vor, mit der nachgewiesen wurde, dass auch der Einsatz von Low-Power Field Programmable Gate Arrays (FPGAs) im IoT für die Ausführung von DNNs möglich ist. Dabei ergaben Messungen von LOTTA auf einem Low-Power FPGA mit externen, nicht-flüchtigen Speichern für die Gewichte eines exemplarischen TCNs eine Leistungsaufnahme von lediglich 27,81 mW, womit das System zum Teil deutlich unter dem Energiebedarf handelsüblicher Low-Power Mikrocontroller liegt.

Abstract

Embedded systems can be found in many application areas, such as robotics or the automotive industry, in which a large number of sensors are integrated. These systems often generate large amounts of data, which must be transferred from the near-sensor platforms to the central control unit in the shortest possible time. Consequently, such platforms require high link bandwidths for the connections between the components used. The increasing use of artificial intelligence (AI) in many of these applications also places further demands on the overall system. In particular, safety-critical applications such as autonomous driving require high data throughput and low latency to process computationally intensive deep neural networks (DNNs). As a result, various analog processing-in-memory (PIM) architectures and purely digital hardware accelerators have been developed over the past decade, which are optimized for the data flow of DNNs and therefore offer better performance and energy efficiency than standard CPUs and GPUs. However, state-of-the-art DNN architectures might consist of several million parameters, which have to be stored as close as possible to the processing unit or the dedicated accelerator for energy- and time-efficient inference. However, the available area, especially of microprocessors close to the sensor, is usually very limited, which is why the internal memory of these platforms often does not meet the requirements of entire DNNs.

One approach to solving these challenges in embedded systems with several hardware accelerators, some of which are distributed, is the inference partitioning of DNNs. This involves distributing the workload across different accelerators in the system in order to achieve higher performance and energy efficiency while simultaneously distributing the area required for the hardware resources across multiple platforms. However, the partitioning of DNNs in embedded systems is a complex task, as various metrics, such as the bandwidth required to transfer intermediate results, must be taken into account. Different approaches exist in the current state of the art, but these often neglect relevant metrics or are only tailored to specific computer architectures.

This thesis therefore presents several novel methods for optimizing such systems at different design levels. These include a generic system-level approach for the inter-layer inference partitioning of DNNs with a focus on Convolutional Neural Networks (CNNs), which uses a variety of non-functional metrics as well as the accuracy of the CNN as a functional metric for optimization. The latter is necessary because hardware accelerators usually do not process floating point data, but mostly fixed point or even integers. The quantization of weights and activations usually leads to a deterioration in accuracy, which is why this metric must also be considered when searching for an optimized partitioning scheme. The CNNParted framework, which implements this methodology, automatically explores the design space of inference partitioning in systems with multiple accelerators, taking into account the available hardware resources. It uses models of the integrated analog PIM and digital hardware accelerators as well as of the links between the platforms in the system to evaluate valid partitioning schemes. Finally, CNNParted generates a Pareto front regarding the relevant metrics from the results of the explorations. The experiments carried out proved that resource-based inference partitioning is advantageous compared to state-of-the-art methods. In some cases, significant improvements in various metrics were achieved for heterogeneous systems consisting of analog PIM and digital hardware accelerators. Consequently, by choosing a Pareto-optimal partitioning scheme, for example, significantly higher energy efficiency is achieved than for running inference on a single accelerator platform.

However, this work also shows that inference partitioning does not always provide benefits without additional optimizations. For example, fulfilling system requirements is usually not guaranteed if low-bandwidth links are used to transfer intermediate results. Consequently, the software must be optimized to enable inference partitioning. At the same time, however, due to the limited energy budget in applications such as the Internet of Things (IoT), the hardware must also be optimized in order to infer DNNs. Commercially available microcontrollers usually do not have the necessary performance and energy efficiency to perform inference. As there are only a few methods in the current state of the art that carry out a corresponding hardware/software co-design, this thesis shows further approaches to solving these problems. For example, by implementing approximated multipliers and an approximated activation function, a very small hardware accelerator can be realized whose architecture leads to only a small loss in the accuracy of the DNN when using Quantization-Aware Training (QAT). Furthermore, this paper presents the LOTTA accelerator architecture for Temporal Convolutional Networks (TCNs), which proves that the use of low-power FPGAs in the IoT is also possible for

the execution of DNNs. Measurements of LOTTA on a low-power FPGA with external non-volatile memories for the weights of an exemplary TCN resulted in a power consumption of only 27,81 mW, which means that the system is significantly below the energy requirements of commercially available low-power microcontrollers.

Vorwort

Die vorliegende Dissertation entstand während meiner Zeit als wissenschaftlicher Mitarbeiter am Institut für Technik der Informationsverarbeitung (ITIV) des Karlsruher Instituts für Technologie (KIT). An dieser Stelle möchte ich all jenen meinen herzlichen Dank aussprechen, die mich auf diesem Weg begleitet und unterstützt haben.

Mein besonderer Dank gilt Prof. Jürgen Becker für die Betreuung meines Promotionsvorhabens, sein Vertrauen und die Freiheiten, die er mir im Rahmen meiner Arbeit am Institut gewährt hat. Ebenso danke ich Prof. Mladen Berekovic für die Übernahme des Korreferats und sein Feedback zur inhaltlichen Vervollständigung der Arbeit. Nicht vergessen möchte ich auch die Prüfungskommission, bestehend aus Prof. Mike Barth, Prof. Cornelius Neumann und Prof. Ivan Peric, welchen ich für Ihre Zeit danken möchte.

Darüber hinaus bedanke ich mich herzlich bei meinen ehemaligen Kolleginnen und Kollegen des ITIV, mit denen ich spannende Diskussionen führen und eine gute Zusammenarbeit in Projekten und der Lehre erleben durfte. Sie haben mir eine bereichernde Zeit am Institut ermöglicht und immer wieder neue Wege für meine Forschung aufgezeigt, wovon ich in vielerlei Hinsicht profitieren konnte. Ebenso danke ich allen Studierenden, die ich im Rahmen von Abschlussarbeiten und HiWi-Tätigkeiten betreuen durfte und die zum Gelingen dieser Arbeit beitrugen. Nicht zu vergessen sind die Mitarbeiterinnen und Mitarbeiter aus der Verwaltung und insbesondere dem Sekretariat des ITIV, die mich auf meinem Weg stets unterstützt haben. Ohne all diese Menschen wäre die vorliegende Arbeit nicht möglich gewesen.

Zu guter Letzt möchte ich meinen Freunden und meiner Familie, meinen Eltern, meinem Bruder und meiner Freundin meinen tiefsten Dank aussprechen. Ihre ständige Unterstützung und Motivation haben mich während dieser Zeit stets vorangebracht und mir auch in schwierigen Phasen Halt gegeben.

Karlsruhe, im Februar 2025
Fabian Kreß

Inhaltsverzeichnis

Kapitel 1

Einleitung

Innerhalb der letzten Dekade hat das Interesse an der Entwicklung neuer Algorithmen für Künstliche Intelligenz (KI) in vielen Fachgebieten und Forschungsbereichen stark zugenommen. Dies ist insbesondere auf die Fortschritte im Bereich der künstlichen neuronalen Netze, sogenannte Deep Neural Networks (DNNs), zurückzuführen, welche unter anderem die Basis für das inzwischen weltweit bekannte Tool ChatGPT bilden. In diesem Fall handelt es sich um ein Large Language Model (LLM), welches insbesondere auf große Datenmengen angewiesen ist, um durch langwieriges Training gute Ergebnisse zu liefern [1]. Ähnliche Anforderungen ergeben sich in anderen Bereichen, wie der Bilderkennung, welche beispielsweise in der Medizintechnik zum Einsatz kommt [2]. Für die Inferenz, also die Ausführung dieser KI-Algorithmen, bedarf es hoher Rechenkapazitäten, weshalb hierfür in der Regel leistungsstarke Server genutzt werden.

Abseits dieser Entwicklung existieren jedoch auch DNNs für eingebettete Systeme, welche beispielsweise die Objekterkennung anhand von Kameradaten mit hoher Genauigkeit durchführen. So finden sich DNNs inzwischen nicht nur in Anwendungen im Bereich der Fahrassistenzsysteme [3] wieder, sondern auch im Bereich der Assistenzrobotik [4], zur intelligenten Überwachung von Maschinen [5] oder der Gesundheitsvorsorge [6]. Auf dieser Grundlage erschließen sich viele neue Möglichkeiten zur Automatisierung von Aufgaben [7].

1.1 Motivation

Über die vergangenen Jahre ist dabei ein klarer Trend hin zu immer aufwendiger werdenden Algorithmen zu beobachten [8]. Dabei nimmt nicht nur die Komplexität der Netze zu, sondern insbesondere auch die Anzahl

der Parameter und damit das dafür benötigte Speichervolumen. Dies stellt besonders eingebettete Systeme vor eine enorme Herausforderung, da diese in der Regel limitiert in der Fläche und dem maximalen Energiebedarf sind [9].

Aus diesem Grund wurden in den letzten Jahren verschiedene Hardwareplattformen vorgestellt, welche für unterschiedliche Aufgabenfelder optimiert sind. Dies reicht dabei von kleinen Mikrocontrollern wie dem GAP8 von GreenWaves Technologies [10], über spezialisierte Hardware-Beschleuniger Architekturen wie der Google Edge TPU [11] bis hin zu eingebetteten Plattformen wie der NVIDIA Jetson Architektur, bestehend aus einer Graphics Processing Unit (GPU) und einem zusätzlichen Beschleuniger, dem sogenannten Deep Learning Accelerator (DLA), für KI-Anwendungen [12]. Der GAP8 eignet sich dabei aufgrund der hohen Energieeffizienz sehr gut für den Einsatz nah am Sensor, verfügt dagegen allerdings nicht über genügend Performance für die schnelle Ausführung größerer Netze. Plattformen wie NVIDIA Jetson erzielen sehr geringe Latenzen für die Inferenz, haben jedoch einen deutlich höheren Energiebedarf und eignen sich daher nicht für den Einsatz in jedem Anwendungsfall.

Neben solchen digitalen Plattformen ist in der Forschung zunehmend ein Trend hin zu In-Memory Beschleunigern zu beobachten, welche in situ Rechenoperationen im Speicher ausführen [13, 14]. Die Motivation für diese Entwicklung ist in Abbildung 1.1 dargestellt. Im Vergleich zur Verbesserung der Hardwareperformance von Prozessoren und anwendungsspezifischen Beschleunigern entwickelt sich die verfügbare Speicherbandbreite langsamer, sodass an dieser Stelle ein Flaschenhals entsteht. Diese sogenannte Memory Wall resultiert in einer Begrenzung der Berechnungsdauer datengetriebener Anwendungen nicht durch den Prozessor selbst, sondern insbesondere durch den Speicher. Durch Ausführung von Berechnungen im Speicher steht eine wesentlich höhere Speicherbandbreite zur Verfügung„ was in der Regel zu einer deutlichen Erhöhung der Performance führt, unter anderem für Anwendungen wie die Inferenz von DNNs. Allerdings geht dies meist auf Kosten eines höheren Flächenbedarfs, weshalb der Einsatz solcher Hardwarebeschleuniger nicht in jedem Anwendungsfall sinnvoll ist. Aus diesem Grund bieten heterogene Architekturen, wie in [15, 16] vorgeschlagen, die Möglichkeit, zwei unterschiedliche Hardwarebeschleuniger für die Inferenz von DNNs zu nutzen.

Ähnlich hierzu arbeiten Forschungsgruppen weltweit an Chiplet-basierten Systemen für den Einsatz in leistungsfähigen Systemen [17, 18]. Der Vorteil der Chiplet-Technologie ist eine erhöhte Ausbeute korrekt funktionierender

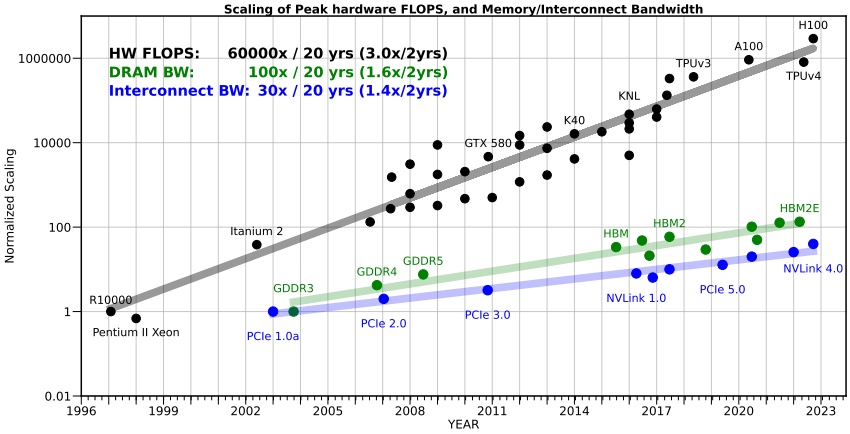


Abbildung 1.1: Zeitliche Entwicklung von Hardware-Performance und Interconnect-/Speicherbandbreite aus [19]

Chips in der Produktion, da das gesamte System nicht auf einem einzigen Die realisiert werden muss. Die hohe Performance des Systems basiert auf der Kombination von leistungsstarken Recheneinheiten auf den Chiplets mit einem effizienten Interconnect, der die verschiedenen Komponenten miteinander verbindet. Im Gegensatz zu den in Abbildung 1.1 betrachteten Links weist eine solche interne Verbindung der Chiplets über einen sogenannten Interposer eine deutlich höhere Bandbreite auf, wodurch der Datenaustausch zwischen verschiedenen Prozessoren und Hardwarebeschleunigern wesentlich schneller erfolgt. Allerdings ist zu berücksichtigen, dass dieser Ansatz eine Verringerung der Flexibilität des Systems zur Folge hat, da bereits zur Entwurfszeit die einzelnen Komponenten festgelegt werden müssen. Dies bedingt gleichzeitig eine Erhöhung der Komplexität durch einen deutlich vergrößerten Entwurfsraum.

1.2 Zielsetzung und Beitrag der Arbeit

In den letzten zehn Jahren haben viele Untersuchungen nachgewiesen, dass die Aufteilung der Inferenz über mehrere Hardwarebeschleuniger vorteilhaft hinsichtlich verschiedener Metriken sein kann (siehe Kapitel 3). Dies gilt neben heterogenen und Chiplet-basierten Plattformen auch für verteilte eingebettete Systeme, bestehend aus mehreren, über einen entsprechenden Link

miteinander verbundenen Recheneinheiten. Die vorliegende Arbeit hat zum Ziel, eine solche Partitionierung für unterschiedliche Systemarchitekturen unter Beachtung unterschiedlicher Anforderungen durchzuführen und dabei eine optimierte Aufteilung der DNN-Inferenz zu erzielen. Hierfür notwendig ist ein Hardware/Software Co-Design, ausgehend von vortrainierten DNNs, welches Anpassungen auf beiden Seiten vornimmt, um den Systemanforderungen gerecht zu werden (siehe Kapitel 4).

Als wesentlicher Bestandteil der Methodik dient dabei das im Rahmen der Arbeit entwickelte Open-Source Framework CNNParted, welches in Kapitel 5 detailliert vorgestellt wird. Für die Suche nach einem optimierten Partitionierungsschema bezieht es dabei die Hardwarearchitektur der einzelnen Systemkomponenten in Form analytischer Modelle ein, um bereits zur Entwurfszeit eine Aufteilung möglicher Anwendungen zu untersuchen. In diese Exploration des Entwurfsraums fließen dabei funktionale und nicht-funktionale Metriken ein, um unterschiedlichen Systemanforderungen zu genügen. Die abschließende Diskussion der Methodik erfolgt im Kontext eines verteilten eingebetteten Systems sowie einer Chiplet-basierten Plattform, um die Flexibilität des CNNParted-Frameworks hinsichtlich unterschiedlicher Systemarchitekturen zu demonstrieren.

Gleichzeitig legt die vorliegende Arbeit aber auch dar, dass eine Aufteilung der DNN-Inferenz nicht in jedem Fall sinnvoll ist und es weiterer Optimierungen der Hard- und Software bedarf, um den Einsatz von KI in eingebetteten Systemen zu ermöglichen. Aus diesem Grund stellt Kapitel 6 unterschiedliche Optimierungen vor, welche von der Quantisierung der Gewichte zur Reduktion des Speicherbedarfs bis zum Entwurf von spezialisierten Hardwarebeschleunigern reichen. Letztere Architekturen nutzten dabei entweder approximative Rechenoperationen, um die benötigten Hardwareressourcen signifikant zu reduzieren, oder zielen auf eine energieeffiziente Nutzung der vorhandenen Logikblöcke auf einem Low-Power Field Programmable Gate Array (FPGA) ab. Abschließend behandelt Kapitel 7 eine Optimierung auf technologischer Ebene, um zusätzliche funktionale Anforderungen an Systeme in sicherheitskritischen Anwendungen zu erfüllen.

Kapitel 2

Grundlagen

Die vorliegende Arbeit behandelt die ressourcenbasierte Inferenz-Partitionierung von DNNs in eingebetteten Systemen, wie diese beispielsweise in Fahrzeugen für Advanced Driver-Assistance System (ADAS) oder auch im Internet-of-Things (IoT) zum Einsatz kommen. Nach Bringmann et al. handelt es sich bei eingebetteten Systemen um *„Rechensysteme, die in einen technischen Kontext, bzw. in ein übergeordnetes System eingebunden sind und vordefinierte Aufgaben erfüllen“* [20]. Sie setzen sich üblicherweise aus Prozessor(en), Speicher, Schnittstellen nach außen und ggf. Rechenbeschleuniger zusammen, in heterogenen Plattformen werden unter anderem auch FPGAs integriert. Dabei vernetzt ein interner Systembus die einzelnen Komponenten, um einen Datenaustausch zu ermöglichen. Aufgrund der geringen verfügbaren Fläche in den genannten Anwendungsfällen kommen dort in der Regel System-on-Chips (SoCs) bzw. Multiprozessor-System-on-Chips (MPSoCs) zum Einsatz, wobei das komplette System auf einem Chip realisiert ist. Im Folgenden werden die zum Verständnis der vorliegenden Arbeit benötigten Konzepte und Begriffe näher erläutert.

2.1 Deep Neural Network (DNN)

KI gilt als Grundlage vieler Anwendungen der Zukunft wie dem autonomen Fahren oder auch der Robotik. Insbesondere das Feld des Machine Learning (ML), welches zum Ziel hat, Algorithmen zu entwerfen, die es einem Computer erlauben, aus Daten zu lernen und entsprechend zu handeln, ist dabei ein zentrales Element der Forschung. Nach Kelleher umfasst ML *„die Entwicklung und Evaluation von Algorithmen die es einem Computer ermöglichen, Funktionen aus einem Datensatz zu extrahieren“* [21]. Dabei entspricht ein Datensatz einer Menge an Beispielen, deren Lösung bekannt ist. Anhand dessen können im Laufe des Trainingsprozesses vom Algorithmus Muster erkannt werden

und diese in eine Funktion eingehen. Innerhalb des Teilgebiets des ML, welches als Deep Learning bezeichnet wird, erfolgt die Darstellung der zuvor genannten Funktionen mittels sogenannter DNNs.

DNNs bestehen aus mehreren unterschiedlichen Schichten (Layern), welche zur Aufgabe haben, jeweils sich wiederholende Merkmale (Features) in den Eingangsdaten zu erlernen und diese entsprechend während der Inferenz zu erkennen. Je nach Komplexität der Aufgabe sind daher unterschiedliche Arten und Strukturen von DNNs erforderlich, um möglichst präzise Ergebnisse und damit eine hohe Genauigkeit der Erkennung eines Musters in den Daten zu erzielen. Die Suche nach solchen Netzen wird dabei als Neural Architecture Search (NAS) bezeichnet. Die nachfolgenden Unterabschnitte stellen die für das Verständnis der vorliegenden Arbeit notwendigen Arten von DNNs und deren Funktionsprinzip vor.

2.1.1 Convolutional Neural Network (CNN)

Für die Verarbeitung strukturierter Daten in gitterförmiger Anordnung wie beispielsweise Bilder, haben bislang besonders Convolutional Neural Networks (CNNs) [22] sehr gute Ergebnisse geliefert. Aus diesem Grund bilden diese bis heute die Grundlage vieler Bildverarbeitungsaufgaben wie z.B. der Objekterkennung, -klassifizierung oder semantischen Segmentierung. Innerhalb der ersten Schichten eines CNNs werden in der Regel Bildmerkmale (Features) mittels Faltungsoperation extrahiert. Am Ende des Netzes erfolgt eine Zusammensetzung dieser Features und schließlich die Ausgabe abhängig vom Anwendungsfall.

Convolutional Layer

Die namensgebende Faltung sorgt in den Convolutional Layer (CONV) dafür, dass ein Bild schrittweise mittels zweidimensionaler Filter auf diese Merkmale abgesucht wird [21]. Dadurch erzielen CNNs recht präzise Ergebnisse mit verhältnismäßig wenigen Gewichten und sind somit gleichzeitig besser skalierbar. Sie ist in der diskreten eindimensionalen Form zu Verarbeitung in der Hardware wie folgt definiert [23].

$$s(t) = (i * w)(t) = \sum_{m=-\infty}^{\infty} i(m) \cdot w(t-m) \tag{2.1}$$

Auf dieser Grundlage leitet sich die Implementierung der Faltungsoperation für die zweidimensionalen Bilddaten im CNN wie folgt ab [24].

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) \cdot K(m,n), \tag{2.2}$$

wobei I den Eingangsdaten bzw. den sogenannten Aktivierungen, und K dem zweidimensionalen Filter, bestehend aus Gewichtungsfaktoren, kurz Gewichten, entspricht. Üblicherweise werden mehrere Filter innerhalb einer Schicht verwendet, wodurch diese Operation entsprechend der Anzahl wiederholt wird. Diesem CONV-Layer folgt häufig eine Pooling-Schicht, um die Aktivierungen zu komprimieren.

Fully-Connected Layer

CNNs können aus lediglich diesen Layern zusammengesetzt werden, um beispielsweise eine semantische Segmentierung vorzunehmen [25]. In diesem Fall werden sie als Fully-Convolutional Networks (FCNs) bezeichnet. Allerdings ist dies in vielen Anwendungen nicht sinnvoll, um eine hohe Genauigkeit zu erzielen. Stattdessen werden wie zuvor beschrieben Fully-Connected Layer am Ende der CNNs verwendet, um beispielsweise eine Klassifizierung vorzunehmen. Dabei ist intern jedes Eingangs- mit jedem Ausgangsneuron über eine gewichtete Kante verbunden. Die Ausgangsaktivierungen S lassen sich folglich, wie in Gleichung (2.3) beschrieben, über eine Matrixmultiplikation der Eingangsdaten I mit der Gewichtsmatrix W und anschließender Addition des Bias berechnen [26].

$$S = I \cdot W = \sum_{j=1}^{m} i_{ij} \cdot w_{jk} \tag{2.3}$$

2.1.2 Long Short-Term Memory (LSTM)

CNNs verwenden nur die aktuellen Eingabedaten für die Inferenz und speichern nicht die Ergebnisse früherer Durchläufe. Es gibt jedoch Anwendungen, bei denen genau dies von Vorteil ist. Aufgrund dieser Anforderungen wurden Recurrent Neural Networks (RNNs) für die Verarbeitung von sequenziellen Daten entwickelt. Frühe Versionen solcher DNNs litten jedoch erheblich unter verschwindenden Gradienten in der Lernphase, was zu sehr langsamem oder gar unbrauchbarem Training führte. Um dieses Problem zu lösen, schlugen

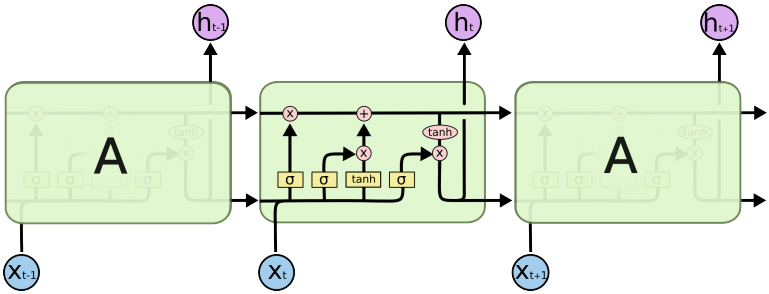


Abbildung 2.1: Schematische Darstellung eines LSTMs aus [28]

Hochreiter und Schmidhuber die sogenannten Long Short-Term Memorys (LSTMs) vor [27].

LSTMs finden sich heute in unterschiedlichen Anwendungen wieder, beispielsweise in der prädiktiven Instandhaltung [29] oder der Spracherkennung [30]. Wie in Abbildung 2.1 dargestellt, besteht ein LSTM aus mehren Zellen, welche miteinander verbunden sind. Diese speichern jeweils ihren eigenen Zustand, wodurch Informationen über frühere Ereignisse in die Inferenz einfließen. Der Zustand wird dabei durch mehrere Layer (Gates) bestimmt, nämlich das Forget- (f_t), Input- (i_t) und Output-Gate (o_t) sowie der Candidate-Speicher ($\tilde{c}_t$). Mathematisch lässt sich eine solche Zelle mit ihren Gates wie folgt beschreiben:

$$i_t = \sigma(W_x^i x_t + W_h^i h_{t-1} + b^i) \tag{2.4}$$

$$f_t = \sigma(W_x^f x_t + W_h^f h_{t-1} + b^f) \tag{2.5}$$

$$o_t = \sigma(W_x^o x_t + W_h^o h_{t-1} + b^o) \tag{2.6}$$

$$\tilde{c}_t = \tanh(W_x^c x_t + W_h^c h_{t-1} + b^c) \tag{2.7}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{2.8}$$

$$h_t = o_t \odot \tanh(c_t) \tag{2.9}$$

wobei $\sigma(x)$ (Sigmoid-Funktion) und $tanh(x)$ die Aktivierungsfunktionen darstellen, die Eingabe zum Zeitpunkt t als x_t und der Hidden-State als h_t bezeichnet wird. Unter Verwendung der Gewichtsmatrix W sowie der Biases b berechnet die Zelle den neuen Zellzustand c_t. Somit ergibt sich aus der Eingabe x_t eine Ausgabe der Zelle in Form des Hidden-States h_t, welche auch an die nächste Zelle im LSTM weitergegeben wird.

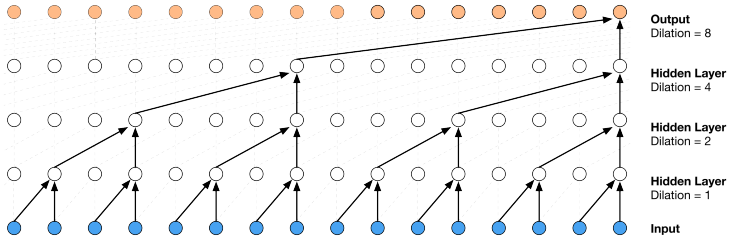


Abbildung 2.2: Schematische Darstellung eines TCN-Stapels aus [31]

2.1.3 Temporal Convolutional Network (TCN)

Eine etwas neuere Form der Verarbeitung von sequenziellen Daten stellen Temporal Convolutional Networks (TCNs) dar, vorgestellt von van den Oord et al. in [31]. Sie sind eine spezielle Art von CNNs und basieren auf gedehnten (dilated) Faltungen mit residualen Blöcken, die in der zeitlichen Dimension angewendet werden. Aufgrund dieser Struktur sind TCNs schneller und einfacher zu trainieren. Zudem weisen sie eine geringere Anfälligkeit für verschwindende Gradienten im Training auf als RNNs, insbesondere im Falle langer Sequenzen [32]. Abbildung 2.2 zeigt beispielhaft den Aufbau eines solchen TCNs.

Gedehnte Faltungen in Kombination mit der Stapelung mehrerer Schichten ermöglichen ein breites rezeptives Feld, das wiederum das Lernen von lang anhaltenden zeitlichen Verbindungen in den Eingabedaten ermöglicht. Dieses rezeptive Feld definiert, wie viele Werte der Eingabe berücksichtigt werden, um eine neue Ausgabe zu erzeugen und lässt sich mathematisch wie folgt formulieren [33]:

$$r = 1 + (k-1) \cdot n_{stack} \cdot \sum_i d_i, \tag{2.10}$$

wobei k der Kernelgröße, n_{stack} der Anzahl der Stapel und d einem Vektor bestehend aus Dehnungsfaktoren (Dilations) innerhalb des Stapels entsprechen. Ist die Länge der Eingabesequenz kleiner als das rezeptive Feld, füllt das TCN die fehlenden Stellen mit Nullen auf. Dies hat im Training allerdings einen negativen Einfluss und beeinflusst damit die Genauigkeit des Netzes. Daher ist bei der Wahl der Hyperparameter zu berücksichtigen, dass das rezeptive Feld eine Größe aufweist, die nicht wesentlich größer ist als die minimale Länge einer Sequenz.

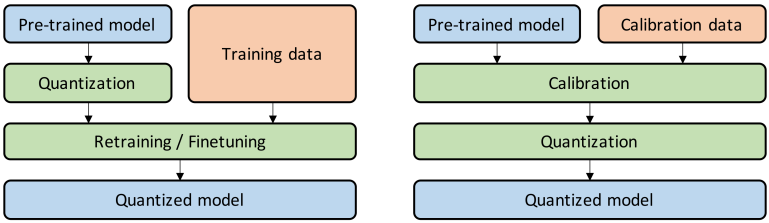


Abbildung 2.3: Vergleich QAT und PTQ aus [35]

2.1.4 Quantisierung

Das Training und die Inferenz von DNNs erfolgt im Rahmen der DNN-Frameworks wie Caffe, Keras oder PyTorch basierend auf Gleitkommazahlen. Im Vergleich zu Festkomma- oder Ganzzahlen sind Rechenoperationen mit Daten in diesem Format aufwendiger und benötigen daher mehr Zeit oder Energie [34]. Um die Inferenz in Hardware zu optimieren, erfolgt in der Regel eine Quantisierung der Gewichte und Aktivierungen. Ist der Trainingsdatensatz bekannt, kann dies auf zwei Arten erfolgen [35], wie in Abbildung 2.3 gezeigt.

Einerseits kann Quantization-aware Training (QAT) angewandt werden, wobei die Quantisierung bereits im Training Berücksichtigung findet. Problematisch ist dabei allerdings, dass die Anpassung der Gewichte auf Gradienten basiert (Backpropagation), welche durch die nicht-differenzierbare Quantisierung meist dem Wert null entsprächen. Daher kommt hierfür häufig der Straight-Through-Estimator (STE) zum Einsatz [36], sodass das Training weiterhin auf Gleitkommazahlen basiert. Im Gegensatz dazu beinhaltet Post Training Quantization (PTQ) kein Retraining des DNNs, wodurch dieses Vorgehen deutlich schneller ist. Lediglich eine Kalibrierung ist notwendig, um die Clipping Range für die Abbildung der Gleitkommazahlen in den entsprechenden Zahlenraum festzulegen. Dafür werden mehrere Inferenzen mit Testdaten durchgeführt und der Bereich zwischen minimalem und maximalem Wert einer Aktivierung ermittelt. Im Vergleich zur symmetrischen Quantisierung führt die asymmetrische Quantisierung in der Regel zu einer deutlich kleineren Clipping Range, was eine geringe Schritthöhe zwischen den Quantisierungsstufen zur Folge hat. [35]. Die Anwendung von PTQ ohne zusätzliche Optimierung führt jedoch üblicherweise zu einer signifikanten Verringerung der Genauigkeit des Netzes [37].

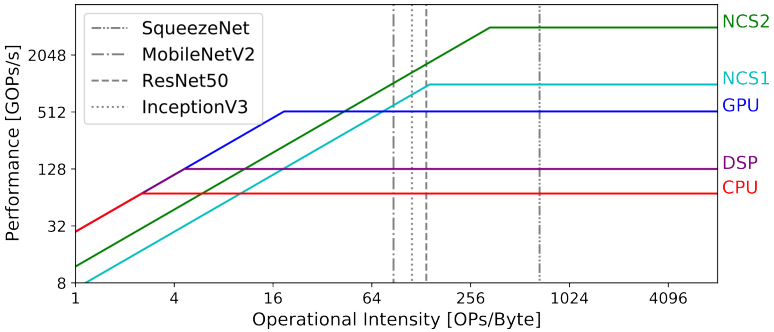


Abbildung 2.4: Roofline Model für verschiedene Prozessoren und CNNs aus [38]

2.2 DNN Hardwarebeschleuniger

Um DNNs effizient in eingebetteten Systemen hinsichtlich Latenz und Energiebedarf zu integrieren, bedarf es leistungsfähiger Hardwareplattformen. Insbesondere für sicherheitskritische Anwendungen müssen diese Systeme Garantien erbringen, die nicht mit einer herkömmlichen Central Processing Unit (CPU) zu erzielen sind. Dies zeigt auch Abbildung 2.4, welches verschiedene Plattformen hinsichtlich der Performance untersucht. Die Darstellung mittels *Roofline Model* ermöglicht einen grafischen Vergleich der jeweiligen maximalen Rechenleistung (horizontale Linien) und maximalen Speicherbandbreite (diagonale Linien) der Systeme. In Bezug auf die vier untersuchten CNN-Anwendungen lässt sich feststellen, dass die maximalen Rechenleistungen der Plattformen eine limitierende Wirkung auf die Leistungsfähigkeit dieser Anwendungen haben. Dies trifft gleichermaßen auf CPU, GPU und Digitaler Signal-Prozessor (DSP) zu. Dagegen ergibt sich für die beiden Varianten des Intel Neural Compute Sticks (NCS) eine Limitierung durch die begrenzte Speicherbandbreite. Mit anderen Worten bedeutet dies, dass die CNNs nicht in der Lage sind, die volle Rechenleistung des Systems zu nutzen, da die Speicheranbindung zu langsam ist. Diese Beobachtung von Cao et al. [38] zeigt ähnlich wie die Arbeit von Zhang et al. [39] auf, dass DNNs während der Inferenz viele Daten laden und speichern müssen. Daher sind optimierte Hardwarearchitekturen erforderlich, die eine hohe Rechenleistung und Speicherbandbreite zur Verfügung stellen. Eine Übersicht des aktuellen Stands der Technik ist beispielsweise in [Guo] gegeben. Nachfolgend wird jedoch nur auf die für die vorliegende Arbeit relevanten Architekturen kurz eingegangen.

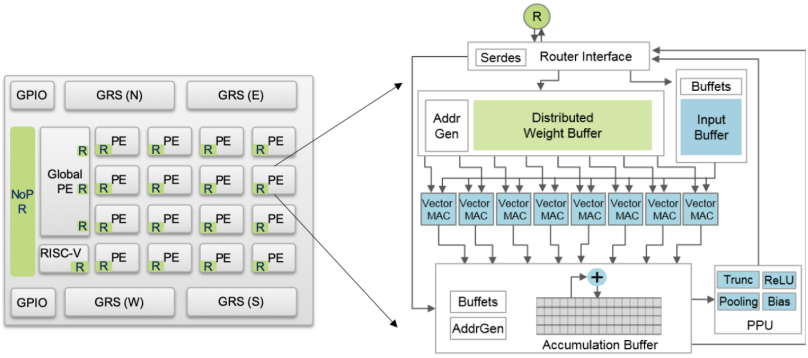


Abbildung 2.5: Simba-Chiplet Architektur aus [40]

2.2.1 Vektorprozessoren

Eine den herkömmlichen CPUs und GPUs ähnliche Architektur ist die Integration parallel Vektoreinheiten zur Berechnung von Multiply-Accumulate (MAC)-Operationen. Wie im Beispiel der Simba-Architektur [40] zu sehen (s. Abbildung 2.5), werden dafür zunächst die Gewichte aus dem Speicher geladen und anschließend mathematische Operationen zusammen mit verschiedenen Eingangsdaten durchgeführt. Die Vektoreinheiten berechnen dabei parallel das Skalarprodukt und geben das Ergebnis weiter. Durch die Integration eines lokalen Puffers für die Gewichte und einer direkte Anbindung an die MAC-Einheiten wird einerseits die Speicherbandbreite erhöht und andererseits werden durch die Wiederverwendung der Gewichte über mehrere Takte die Anforderungen an diese verringert. Diese Architektur erlaubt insgesamt eine hohe Flexibilität hinsichtlich des Datenflusses und ist damit auch für große DNNs gut geeignet [40].

2.2.2 Systolische Arrays

Bei der Untersuchung von Hardware-Beschleunigern für DNNs ist festzustellen, dass der Aufbau eines systolischen Arrays eine häufig anzutreffende Architektur darstellt. Dies ist insbesondere in eingebetteten Systemen der Fall, bei denen sowohl ein geringer Stromverbrauch als auch eine hohe Performance gefordert sind. Eine solche Architektur ist in Abbildung 2.6 beispielhaft anhand von Gemmini [41] dargestellt. Die zweidimensionale

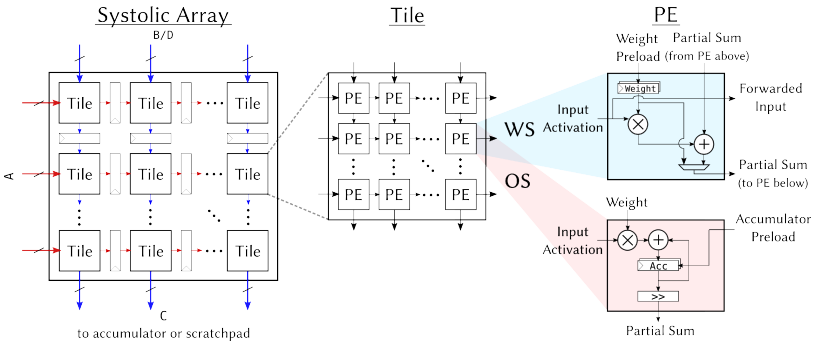
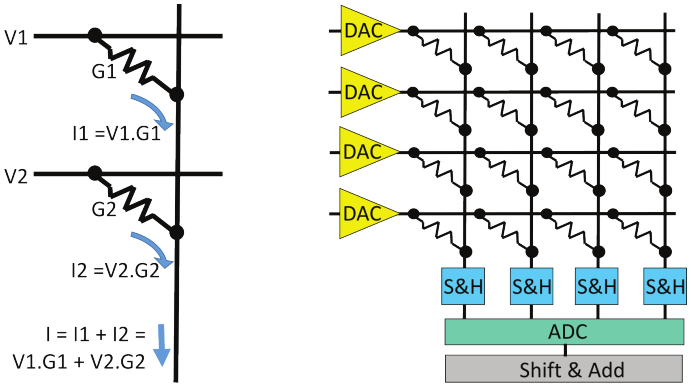


Abbildung 2.6: Gemmini Architektur aus [41]

Struktur ermöglicht die Weitergabe von (Zwischen-)Ergebnissen in zwei Richtungen innerhalb eines Takts, sodass Daten schnell weiterverarbeitet werden. Durch eine solch stark datenflussgetriebene Architektur erzielen derartige Hardwarebeschleuniger einen hohen Grad an Datenwiederverwendung und kommen daher gut mit einer geringeren Speicherbandbreite aus. Nachteilig an dieser Architektur ist die teilweise geringe Auslastung der Recheneinheiten im Array, da ein Layer nicht immer exakt passend auf die Hardware abgebildet werden kann. Dies ist besonders für in der Fläche beschränkte Systeme von Bedeutung.

Grundsätzlich ergeben sich für solche Architekturen mehrere mögliche Varianten des Datenflusses [42]. Im Falle von Weight-Stationary werden die Gewichte im lokalen Register der MAC-Einheit gespeichert und für mehrere Eingangsdaten wiederverwendet [41, 43]. Folglich wandern Eingangsaktivierungen und Zwischenergebnisse durch das systolische Array. Dagegen übertragen Output-Stationary Architekturen die Aktivierungen und Gewichte innerhalb des Arrays und speichern die Zwischenergebnisse im internen Speicher, bis die Berechnung abgeschlossen ist [44, 45]. Der Row-Stationary-Datenfluss basiert auf der Berechnung einer Reihe der Faltung in einer MAC-Einheit, wobei die Gewichte in einem internen Registersatz gespeichert werden [46, 47]. Dadurch müssen weder Zwischenergebnisse noch Gewichte innerhalb des systolischen Arrays weitergegeben werden. Abschließend ist auch eine Realisierung ohne Register in den MAC-Einheiten möglich (No Local Reuse). Diese Architekturen zielen auf einen minimierten Flächenbedarf ab und nehmen den dadurch den gestiegenen Bedarf an eine hohe Bandbreite hin [48]. Je nach Systemanforderungen wird somit ein entsprechender Aufbau des systolischen Arrays ausgewählt.



(a) Multiply-Accumulate operation (b) Vector-Matrix Multiplier

Abbildung 2.7: RRAM-Crossbar für die Ausführung von MAC-Operationen aus [49]

2.2.3 Processing-In-Memory

Für die Beschleunigung von DNNs in der Hardware gewinnen analoge Processing-In-Memory (PIM)-Architekturen zunehmend an Bedeutung. Basierend auf neuartigen Speichertechnologien wie Magnetoresistive Random Access Memory (MRAM), Resistive Random Access Memory (RRAM), oder Phase Change Random Access Memory (PCM) für diese eine Matrix-Vector-Multiplikation (MVM) direkt im Speicher aus, wodurch die Notwendigkeit der Übertragung von Gewichten entfällt. Im Vergleich zu anderen Speichertechnologien bieten sie eine hohe Dichte, inhärente Nichtflüchtigkeit und einstellbare Widerstandseigenschaften. Durch die zweidimensionale Speichermatrix aus solchen Bausteinen, die sogenannte Crossbar, werden die Berechnungen direkt analog ausgeführt, wie in Abbildung 2.7 schematisch dargestellt. Anhand des Stroms oder der Spannung auf den Bitleitungen lassen sich schließlich mittels Analog-to-Digital Converters (ADCs) die Ergebnisse der MVM auslesen. Da diese Operation die Grundlage für CONV- und Fully-Connected-Layer bilden, erlaubt die Verwendung von analogen PIM-Komponenten im System eine deutliche Steigerung der Leistungsfähigkeit und der Energieeffizienz im Vergleich zu Complementary Metal Oxide Semiconductor (CMOS)-basierten Beschleunigern [50]. Ein Nachteil ist allerdings die hohe Fehleranfälligkeit durch Überlagerung von Rauschen auf den Leitungen und Ungenauigkeiten im ADC bzw. den Bitzellen in der Crossbar. Folglich existieren verschiedene Techniken, die eine Fehlerkompensation in PIM-basierten Beschleunigern ermöglichen,

wie beispielsweise von Eldebiky et al. [51] vorgeschlagen durch das Einfügen entsprechender lernfähiger Layer in das DNN.

2.2.4 Hardware-Mapping

Die zeit- und energieeffiziente Inferenz eines DNNs in einem Hardwarebeschleuniger hängt maßgeblich von der Abbildung des Datenflusses auf die Architektur ab. Ein schlechtes Mapping führt demnach zu einer schlechten Auslastung der vorhandenen Ressourcen und folglich zu einer verringerten Performance und Energieeffizienz. Zusätzlich ist das Ziel eines guten Mappings ein hoher Grad an Wiederverwendung von geladenen Daten, um die Anzahl der zeit- und energieintensiven Speicherzugriffe zu reduzieren [52]. Dies kann zum einen auf räumliche Art geschehen, wobei Daten zwischen Recheneinheiten ausgetauscht werden. Eine solche Form der Datenwiederverwendung nutzen unter anderem systolische Arrays explizit aus. Alternativ kann eine Wiederverwendung von Daten auch in der zeitlichen Dimension erfolgen, wie es PIM-Architekturen inhärent implementieren. In diesem Fall werden die Daten, wie unter anderem die Gewichte, bereits vor der Inferenz vollständig lokal in den Beschleuniger geladen.

Wie in Abbildung 2.8 dargestellt, können jedoch schon zuvor Optimierungen des Datenflusses in Abhängigkeit von der Hardwarearchitektur vorgenommen werden. Zum Beispiel führt die Neuordnung der Schleifen eines Layers oder die Aufteilung einer Schleife in mehrere Blöcke zu einem Datenfluss, der sich besser auf die Architektur des Hardwarebeschleunigers abbilden lässt. Diese Optimierung kann dabei auch über Layergrenzen hinweg durchgeführt werden, um eine Datenwiederverwendung zwischen mehreren aufeinanderfolgenden CONVs zu ermöglichen [53].

Sind darüber hinaus die Gewichte bekannt, ist eine weitere Optimierung des Datenflusses möglich. Im Falle von Nullgewichten ist das Ergebnis der Multiplikationsoperation, die häufig in DNNs verwendet wird, bereits bekannt und die Berechnung kann daher entfallen. Darüber hinaus haben viele Gewichte einen Wert nahe Null, sodass die Multiplikation mit der Eingangsaktivierung keinen relevanten Einfluss auf die Inferenz hat. An dieser Stelle kann das sogenannte Pruning eingesetzt werden, welches alle Werte der Gewichte unterhalb einer definierten Schwelle auf Null setzt und somit eine Ausführung der mathematischen Operation überflüssig macht [54]. Daher ist auch dieser Einfluss auf den Datenfluss und damit auf das Mapping des Layers auf den Hardwarebeschleuniger zu berücksichtigen.

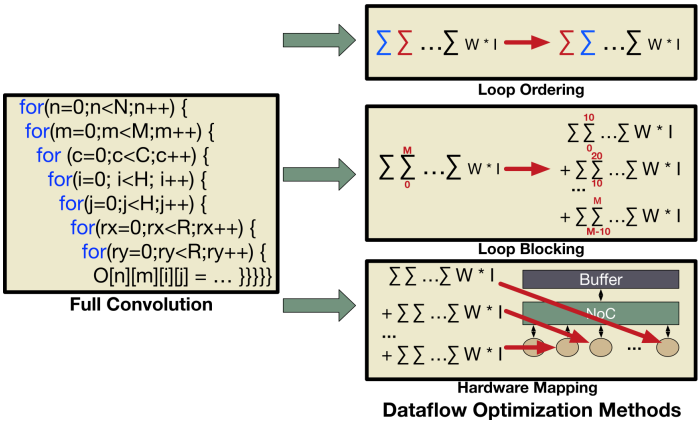


Abbildung 2.8: Optimierung des DNN-Datenflusses für das HW-Mapping aus [53]

2.3 Mehrzieloptimierung

Die Realisierung einer ressourcenbasierten Inferenz-Partitionierung in eingebetteten Systemen setzt die Betrachtung mehrere Metriken im Hardware/-Software Co-Design voraus. Aus diesem Grund handelt es sich hierbei um ein Mehrzieloptimierungsproblem, zu dessen Lösung sich in der Literatur verschiedene Verfahren finden lassen [55]. Beispielsweise sortiert die lexikographische Optimierung die Ziele nach abnehmender Wichtigkeit und optimiert anschließend schrittweise das Problem ausgehend vom wichtigsten Ziel. Für die in der vorliegenden Arbeit durchgeführte Optimierung des Systems ist dies allerdings von Nachteil. Ergibt sich bereits im ersten Schritt des Verfahrens eine einzelne optimale Lösung, werden nachfolgende Ziel nicht mehr betrachtet. Findet der Algorithmus beispielsweise zunächst eine einzige minimale Konfiguration hinsichtlich der Latenz, endet die Suche sofort ohne Analyse anderer relevanter Metriken.

Eine weitere Option ist die Gewichtung der einzelnen Zielfunktionen und eine Zusammenführung dieser in Form einer Summe. Auf diese Weise lässt sich das Problem als ganzzahliges lineares Optimierungsproblem darstellen und damit unter anderem durch den Simplex-Algorithmus lösen [56]. Dies ist für die in der vorliegenden Arbeit betrachtete Problemstellung aus zwei Gründen von Nachteil. Einerseits ist die gefundene Lösung stets ein Eckpunkt des zulässigen Bereichs und stellt damit nicht unbedingt die optimale Lösung dar. Wesentlich problematischer ist allerdings die Wahl passender Zielgewichte,

welche in komplexen Systemen unter Beachtung der verschiedenen funktionalen und nicht-funktionalen Metriken nur sehr schwer oder gar nicht in allgemeiner Form zu bestimmen sind. Dies liegt an den zum Teil konkurrierenden Zielen, wodurch eine vorteilhafte Abwägung zwischen den Metriken gefunden werden muss. So führt ein hoher Datendurchsatz durch parallele Verarbeitung von Daten in der Regel zu einem höheren Flächenbedarf im SoC im Vergleich zur sequentiellen Prozessierung der Daten.

Aus diesem Grund verfolgt die vorliegende Arbeit im weiteren Verlauf die Bestimmung möglichst vieler effizienter bzw. Pareto-optimaler Punkte im Suchraum. Solche Punkte sind wie folgt definiert [57].

Definition 1 (Pareto-optimal)**.** *Ein Element $x \in X$ heißt Pareto-optimal bezüglich der Zielfunktionen $c_i, i = 1, \ldots, k$, wenn es kein anderes Element $x' \in X$ gibt, das die folgenden beiden Bedingungen erfüllt:*

$$\forall i, 1 \leq i \leq k : \qquad f_i(x') \geq f_i(x), \tag{2.11}$$

$$\exists i, 1 \leq i \leq k : \qquad f_i(x') > f_i(x). \tag{2.12}$$

Sind Gleichung (2.11) und Gleichung (2.12) erfüllt, dominiert x' das Element x streng und heißt Pareto-optimal. Die Bestimmung der Pareto-Front und damit von entsprechenden Lösungen solcher Mehrzieloptimierungsproblemen ist eine $\mathcal{NP}$-schwere Aufgabe [56] und folglich mit exponentiellem Rechenaufwand verbunden. Daher werden üblicherweise Heuristiken angewandt. Diese können zwar nicht gewährleisten, der Pareto-Front nahe zu kommen, liefern jedoch gute Ergebnisse in annehmbarer Zeit. Viele Metaheuristiken wie Simulated Annealing oder Tabu Search nutzen eine Zielfunktion, welche entsprechend zu optimieren ist.

Für die in dieser Arbeit betrachteten Problemstellungen stellt der genetische Algorithmus mit nicht-dominierter Sortierung (NSGA) [58] und insbesondere dessen optimierte Variante NSGA-II nach Deb et al. [59] ein vorteilhaftes Suchverfahren dar. Allgemein basieren genetischen Algorithmen neben der Evaluation der potenziellen Lösungen auf mehreren Operationen, welche für jede neue Generation durchlaufen werden. Aus einer initial erzeugten Population, also einer Menge an Lösungen, werden zunächst durch Kreuzung der Individuen neue Lösungen generiert. Im Anschluss erfolgt eine zufällige Veränderung des Lösungsvektors durch die Mutation sowie eine Aussortierung einiger weniger geeigneter und nach Gleichung (2.11) dominierter Lösungen, welche durch die Selektion identifiziert werden. Auf diese Weise erkundet der Algorithmus den Suchraum parallel in verschiedene Richtungen hin zur

Pareto-Front. Da es sich bei diesen Verfahren um eine Verbesserungheuristik [56] handelt, ist der Erfolg der Suche jedoch abhängig von der initialen Population. Bei Generierung einer schlechten initialen Population, findet der Algorithmus keine oder nur schlechte Lösungen. Folglich ist es bei der Verwendung solcher Verfahren notwendig, über Konstruktionsheuristiken wie dem Greedy-Verfahren [56] erste gute Lösungen als Ausgangspunkt der weiteren Exploration zu finden.

Kapitel 3

Stand der Technik

Die Inferenz-Partitionierung von DNNs ist ein Forschungsgebiet, das in den letzten Jahren zunehmend an Bedeutung gewonnen hat. Folglich wurden seither verschiedenste Methoden präsentiert, welche die Aufteilung des DNNs abhängig vom Anwendungsfall optimieren. Im Gegensatz zu klassischen Ansätzen, welche die Arbeitslast über das System verteilen indem Funktionen innerhalb der Firmware ausgelagert werden, basiert die Inferenz-Partitionierung insbesondere auf dem Datenfluss. Allgemein lassen sich die Ansätze in Intra-Layer-Partitionierung, also einer parallelen Ausführung von Teilen eines Layers auf verschiedenen Plattformen, sowie Inter-Layer-Partitionierung, einer sequentiellen Verarbeitung der DNN-Schichten, kategorisieren. Dieses Kapitel gewährt einen Einblick in den aktuellen Stand der Technik und zeigt schließlich auf, warum ein neuartiger Ansatz notwendig ist, um eine optimierte Inferenz-Partitionierung unter Beachtung relevanter Systemanforderungen zu erzielen.

3.1 Intra-Layer-Inferenz-Partitionierung

Verteilte eingebettete Systeme können besonders im IoT-Bereich stark von der Aufteilung der Arbeitslast über mehrere Plattformen profitieren. Die Intra-Layer-Inferenz-Partitionierung bietet sich dabei für Systeme an, welche aus vielen kleinen Recheneinheiten bestehen, welche parallel identische Rechenoperationen ausführen. Verfügen diese Plattformen über eine unzureichende Speicherkapazität oder Performance, so ist die vollständige Ausführung eines DNNs oder sogar einzelner Layer nicht möglich. In diesem Fall schafft die Partitionierung eines Layers Abhilfe.

Der aktuelle Stand der Technik umfasst verschiedene Verfahren, welche sich mit diesem Problem auseinandersetzen. Es existieren Ansätze, welche auf

eine Verteilung der Eingangsdaten über verschiedene Clients im Systeme zurückgreifen [60, 61, 62]. Dadurch übernimmt jedes Gerät im System einen kleinen Teil der Berechnung, deren Teilergebnisse anschließend an einem Punkt zusammengeführt werden. Gleichzeitig kann eine explizite Aufteilung der Rohdaten vom Sensor zu den Hardwareeinheiten zusätzlich für einen erhöhten Datenschutz sorgen [63].

Allgemein können die Eingangsdaten der Layer auf unterschiedliche Weise aufgeteilt werden. So schlagen Dey et al. [64] eine Aufteilung der Aktivierungen und der Filter entlang der Tiefe vor. Im Gegensatz zur ebenfalls häufig verwendeten zeilen- oder spaltenbasierten Partitionierung der Daten entsteht dadurch eine geringe Anzahl zusätzlich zu übertragender Informationen, verursacht durch den Stride-Parameter eines Layers. All diese Verfahren haben jeweils verschiedene Vor- und Nachteile und werden daher entsprechend der Anwendungsanforderungen eingesetzt [65, 66]. Die genannten Ansätze zeigen jedoch in ihrer Ausführung über das gesamte DNN betrachtet eine geringe Energieeffizienz sowie Performance. Grund hierfür ist die Notwendigkeit, Zwischenergebnisse zwischen den Recheneinheiten auszutauschen und die Aktivierungen des letzten partitionierten Layers zusammenzuführen. Eine Alternative hierzu ist die Betrachtung der statischen Aufteilung der Schichten bereits während des Trainings. So zeigen unter anderem die Untersuchungen [67, 68], dass dies zu einer signifikanten Reduktion des Datenaustauschs zwischen den Recheneinheiten im System führt. Allerdings bedeutet dies einen zusätzlichen Entwicklungsschritt, welcher den Prozess der NAS deutlich verlangsamt.

Um die Problematik der Datenabhängigkeiten zwischen den Schichten für die Intra-Layer-Inferenz-Partitionierung zu umgehen, wurde das Konzept der sogenannten Layer Fusion entwickelt. Bei diesem Verfahren erfolgt die Aufteilung der Berechnungen derart, dass die Ergebnisse direkt als Eingabe des nächsten Layers verwendet werden. Somit entfällt der Datenaustausch zwischen den Recheneinheiten im System. Hierfür werden die Eingangsaktivierungen je nach Methodik entweder entlang einer [69, 70, 71, 72] oder zweier Achsen [73, 74] aufgeteilt.

Letzteres nutzt auch das Open-Source Framework DeepThings [75], welches als eines der Vorreiter in diesem Forschungsfeld gilt. Eine Übersicht des Aufbaus ist in Abbildung 3.1 dargestellt. Im Zentrum der Arbeit steht das Fused Tile Partitioning (FTP), welches CONV-Layer in unabhängige Segmente partitioniert, die anschließend dynamisch zur Laufzeit über die im System verfügbaren Geräte verteilt werden. Dabei betrachtet das Framework insbesondere frühe CONV-Schichten des CNNs mit geringeren Speicheranforde-

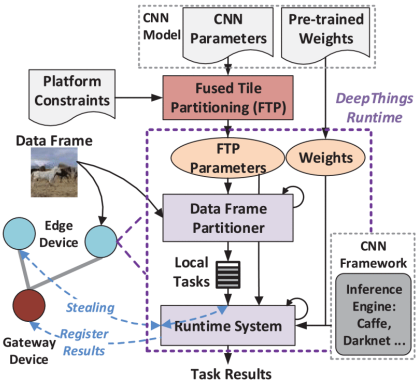


Abbildung 3.1: Übersicht des DeepThings Frameworks aus [75]

rungen, um auch kleine Recheneinheiten einzubinden. Diese Verteilung der Aufgaben an die Plattformen erfolgt in DeepThings unter Beachtung der Leistungsfähigkeit sowie der zuletzt verarbeiteten Segmente, um eine möglichst hohe Datenwiederverwendung und damit einen geringen Kommunikationsaufwand zu erzielen. Auf diese Weise erlaubt die Interaktion zwischen den IoT-Plattformen die Ausführung komplexer CNNs im System mit geringerer Latenz und höherem Datendurchsatz als dies bei der Verwendung eines einzelnen Geräts für die Inferenz der Fall wäre. Abgesehen davon vernachlässigt das Framework jedoch wichtige Metriken wie beispielsweise die Energieeffizienz. Die Optimierung hat demnach ausschließlich das Ziel, eine möglichst hohe Performance zu erreichen.

Bei diesen Verfahren ist es jedoch unvermeidlich, die gleiche Berechnung mehrmals durchzuführen, wenn der Kommunikationsaufwand reduziert werden soll. Je mehr sequenzielle Layer eine Plattform ausführen soll, desto mehr Redundanz ergibt sich im Gesamtsystem. Im aktuellen Stand der Technik existieren Ansätze, welche sich ähnlich zu DeepThings auf die ersten Layer eines Netzes fokussieren und dort einen Austausch von Zwischenergebnissen unterbinden [76, 77]. Diese wenden stattdessen Zero-Padding an, wobei die fehlenden Datenblöcke durch Nullen aufgefüllt werden. Ohne zusätzliches Training führt ein solches Verfahren allerdings zu einer Verringerung der Genauigkeit des CNNs.

Des Weiteren ist bei der Intra-Layer-Inferenz-Partitionierung die Leistungsfähigkeit der verfügbaren Hardware zu berücksichtigen, wodurch die Suche nach einer optimalen Aufteilung an Komplexität zunimmt. Findet diese Me-

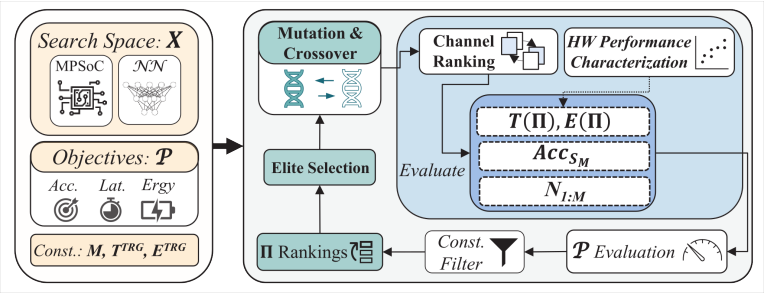


Abbildung 3.2: Übersicht des Map-and-Conquer Frameworks aus [82]

trik keine Beachtung, führt es wahrscheinlich dazu, dass performante Plattformen viel Zeit im untätigen Zustand verbringen. Auch für dieses Problem existieren geeignete Verfahren, die diese Randbedingung in die Verteilung der Aufgaben einbeziehen und folglich die Größe der Partition derart bestimmen, dass diese proportional zur Rechenleistung ist [77, 78, 79]. Folglich erhöht sich die durchschnittliche Auslastung der Plattformen und damit auch die Performance des Gesamtsystems. Auf FPGAs basierende Systeme, welche inzwischen in zahlreichen Anwendungsbereichen des High-Performance Computing (HPC) zum Einsatz kommen, erhöhen die Komplexität der Inferenz-Partitionierung aufgrund der erhöhten Flexibilität durch die Möglichkeit der Rekonfiguration der Hardware. Hierbei besteht die Herausforderung zusätzlich in der Verringerung des Kommunikationsaufwands zwischen den Plattformen bei Beachtung der limitierten Hardwareressourcen der verwendeten FPGAs [80, 81].

Die Inferenz-Partitionierung innerhalb eines heterogenen MPSoCs betrachtet beispielsweise das Framework Map-and-Conquer [82], welches in Abbildung 3.2 dargestellt ist. Das Verfahren analysiert dabei auch den Einfluss der verschiedenen Quantisierungen in den Hardwareeinheiten und zieht dies in die Bewertung der Partitionierungsschemata ein. Um das Mapping-Problem zu lösen, nutzen Bouzidi et al. einen Genetischer Algorithmus (GA), welcher als Ziel die Minimierung des Genauigkeitsverlusts, der Latenz und des Energiebedarfs hat. Auf diese Weise sucht das Framework nach einer optimalen Aufteilung des DNNs. Für die Ermittlung der Systemperformance eines möglichen Partitionierungsschemas setzt das Framework auf ein Gradient-Boosting-Verfahren. Die ML-basierte Methode durchläuft zunächst ein Training mit einem Datensatz, der verschiedene Layerkonfigurationen umfasst. Im Anschluss wird sie als Teil von Map-and-Conquer zur Abschätzung der

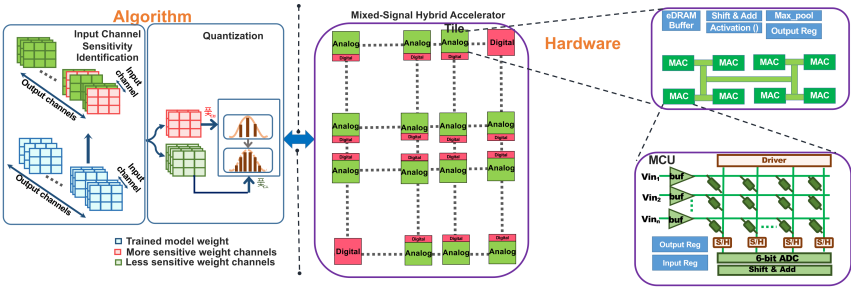


Abbildung 3.3: Übersicht des HybridAC Frameworks aus [84]

Latenz sowie des Energiebedarfs eingesetzt. Basierend auf den Evaluationsergebnissen ermöglicht dieses Verfahren eine signifikante Verbesserung von Energieeffizienz und Latenz der DNN-Inferenz in MPSoCs. Ähnlich diesem Ansatz sucht das ODiMO Framework [83] nach einer vorteilhaften Aufteilung des Netzes auf einen MPSoC bestehend aus analogem und digitalem Hardwarebeschleuniger. Im Gegensatz zum zuvor genannten Verfahren nutzt ODiMO analytische Modelle basierend auf Messergebnissen realer Hardware, um Latenz und Energiebedarf abzuschätzen. Für die Suche nach einem optimalen Partitionierungsschema findet zusätzlich auch der Einfluss der Aufteilung auf die Genauigkeit Berücksichtigung. Dies ist im Falle analoger Beschleuniger von entscheidender Bedeutung, da aufgrund des schlechteren Signal-Rausch-Verhältnisses im Vergleich zu digitalen Architekturen eine höhere Anfälligkeit für Störungen besteht.

HybridAC [84] ermöglicht die Exploration zur Entwurfszeit für hybride Systeme bestehend aus einem analogen und einem digitalen Beschleuniger. Beispielhaft entwickelten die Autoren im Rahmen ihrer Arbeit ein SoC bestehend aus einem großen analogen und einem kleinen digitalen Beschleuniger, wobei letzterer für die Inferenz weniger robuster Kanäle der Layer zuständig ist. Eine Übersicht der Implementierung ist in Abbildung 3.3 dargestellt. Die Auswahl dieser Kanäle erfolgt, indem Gewichte extrahiert werden, die eine geringe Robustheit gegenüber Hardwarefehlern aufweisen. Dafür nutzt HybridAC einen Ansatz basierend auf [85], welcher für die Abschätzung auf die Hesse-Matrix zurückgreift. Durch dieses Hardware/Software Co-Design erreicht HybridAC eine optimierte Partitionierung verschiedener DNNs hinsichtlich relevanter Metriken wie Genauigkeit, Leistungsaufnahme, Fläche und Latenz. Dennoch zielt dieses Framework auf ein System mit dem analogen Beschleuniger im Zentrum ab und erlaubt dadurch keine automatisierte Exploration für Systeme aus mehreren Plattformen.

Aufgrund der hohen Komplexität der Intra-Layer-Inferenz-Partitionierung ist häufig eine umfassende Exploration möglicher Partitionierungsschemata notwendig, um eine optimierte Aufteilung der DNN-Inferenz zu finden. Für leistungsschwache Plattformen, welche über keinen anwendungsspezifischen Hardwarebeschleuniger verfügen, ermöglichen die genannten Verfahren jedoch generell die Ausführung der Inferenz im Gesamtsystem. Daher sind diese Methoden insbesondere für verschiedene eingebettete Systeme im Kontext von IoT relevant.

3.2 Inter-Layer-Inferenz-Partitionierung

Im Gegensatz zu IoT-Anwendungen stehen eingebetteten Plattformen im Automobilbereich oder einem ähnlichen Umfeld wesentlich mehr Ressourcen zur Verfügung. Damit einher geht eine sinkende Bedeutung der Intra-Layer-Inferenz-Partitionierung, da diese in der Regel ineffizient hinsichtlich Performance, Energie- und Speicherbedarf ist. Folglich fokussieren sich im aktuellen Stand der Technik wesentlich mehr Ansätze auf Formen der Inter-Layer-Inferenz-Partitionierung. Die nachfolgenden Ausführungen widmen sich einer vertiefenden Betrachtung der relevanten Aspekte dieser Thematik. Für die genaue Analyse werden dabei zunächst Methoden zur Inter-Layer-Inferenz-Partitionierung für FPGA-basierte und anschließend für eingebettete Systeme vorgestellt.

FPGA-basierte Systeme

Wie zuvor erwähnt, nutzen viele Systeme neben klassischen GPUs, wie unter anderem die von Lane et al. entwickelte Software DeepX [86], inzwischen auch FPGAs für die Inferenz-Partitionierung von DNNs. Dabei kommt der Inter-Layer-Inferenz-Partitionierung ebenfalls eine wichtige Rolle zu, wie verschiedene Studien in den vergangenen Jahren demonstriert haben [87, 88, 89, 90]. Beispielsweise präsentieren Zhang et al. [91] einen Ansatz für das Mapping großer DNNs auf asymmetrische Multi-FPGA-Plattformen unter Beachtung der benötigten Bandbreite zwischen den Partitionen sowie der Ressourcenallokation der jeweiligen FPGAs. Für letztgenanntes werden neben Flip-Flops (FFs) und Look-Up Tables (LUTs) auch DSPs und Block RAMs (BRAMs) betrachtet. Da das Verfahren auf Datencenter zugeschnitten ist, versucht der verwendete Optimierungsalgorithmus ein Partitionierungsschema mit maximalem Datendurchsatz zu finden.

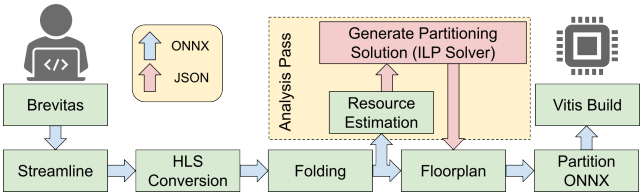


Abbildung 3.4: Übersicht des Elastic-DF Frameworks aus [92]

Ein ähnliches Ziel verfolgt Elastic-DF [92], welches in Abbildung 3.4 dargestellt ist. Das Framework generiert dafür zunächst eine Hardwarerepräsentation des DNNs auf dem FPGA, um daraus eine Abschätzung der benötigten Ressourcen abzuleiten. Unter Beachtung der vorhandenen Blöcke in den jeweiligen FPGAs sowie der Übertragungslatenz zwischen den Partitionen ermittelt das Framework automatisiert eine Aufteilung zwischen den verfügbaren Plattformen. Durch dieses Verfahren erzielt das Framework eine Verringerung der Latenz um bis zu 44 % und einen Anstieg des Durchsatzes um bis zu 78 %.

Kamath et al. [93] schlagen dagegen mit M5 ein Framework für die Inferenz-Partitionierung in FPGA-basierten Systemen vor, welches einen besonderen Schwerpunkt auf die Reduzierung der notwendigen Anzahl an FPGAs im System legt. Zu diesem Zweck nutzt M5 unter anderem einen eigens dafür konzipierten Explorationsalgorithmus, welcher mehr potenzielle Partitionierungsschemata in DNNs mit parallelen Pfaden findet als klassische Scheduling-Algorithmen. Für eine möglichst geringe Latenz und möglichst hohe Auslastung der verwendeten Hardwareressourcen analysiert M5 zusätzlich die Datenabhängigkeiten zwischen den Layern. Basierend auf den Evaluationsergebnissen erreicht die Anwendung des Verfahrens für verschiedene DNNs eine deutliche Reduzierung der benötigten FPGAs.

Eingebettete Systeme

In Bezug auf eingebettete Systeme existiert eine Vielzahl von Verfahren zur Inter-Layer-Inferenz-Partitionierung. Allerdings wurden diese Verfahren bislang vorwiegend zur Untersuchung von im Handel erhältlicher Hardwareplattformen eingesetzt. Einige Methoden betrachten dabei lediglich ein oder zwei Metriken wie Latenz und Energieverbrauch [94, 95, 96, 97]. Beispielsweise evaluiert das von Huang et al. in [98] entwickelte Framework DeePar die Partitionierung zwischen einem mobilen Endgerät, einer Edge-Plattform

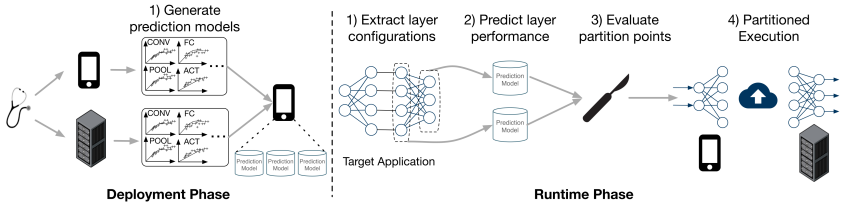


Abbildung 3.5: Übersicht von Neurosurgeon aus [99]

und der Cloud hinsichtlich der Berechnungsdauer. Deren Anwendungsstudie ergibt, dass eine Partitionierung zu einer Reduzierung der Gesamtlatenz von bis zu 80 % führt. Es sei jedoch darauf verwiesen, dass das Framework auf den hier betrachteten Anwendungsfall beschränkt ist und wesentliche Metriken wie den Energiebedarf nicht berücksichtigt.

Als ein Vorreiter der Inter-Layer-Inferenz-Partitionierung von DNNs gilt der Scheduler Neurosurgeon [99], welcher nach einer optimierten Aufteilung der Arbeitslast zwischen mobilem Endgerät und der Cloud sucht. Wie in Abbildung 3.5 gezeigt, teilt sich das Verfahren in zwei Phasen auf. In der Deployment-Phase werden Vorhersagen für verschiedene Layer-Typen für die verwendeten Hardwareplattformen generiert. Diese Informationen werden in der zweiten Phase (Runtime-Phase) genutzt, um dynamisch den besten Partitionierungspunkt zu ermitteln. Dabei werden neben der Latenz auch die Auslastung des Datencenters sowie die geschätzte Leistungsaufnahme für die Inferenz im mobilen Endgerät zur Entscheidungsfindung herangezogen. Daneben zählt auch das Dynamic Adaptive DNN Surgery (DADS) Verfahren nach Hu et al. [100] zu den relevanten Methoden im aktuellen Stand der Technik. Im Vergleich zu Neurosurgeon erlaubt der Graph-basierte Ansatz auch die Partitionierung innerhalb paralleler Pfade, wodurch sich eine deutlich höhere Systemperformance erzielen lässt. Außerdem berücksichtigt DADS den Fall, dass aufgrund anderer Nutzer im Netzwerk nicht die gesamte Bandbreite zur Verfügung steht. Dies beeinflusst die Entscheidung der Partitionierung stark, da ein stark ausgelastetes Netzwerk zu einem deutlichen Anstieg der Latenz für die Übertragung der Daten führt. Gleiches gilt auch für das Framework RoaD-RuNNer [101], welches eine weitere Optimierung im Verfahren verwendet, um eine bessere Skalierbarkeit für große DNNs zu erreichen. So werden nicht alle Layer des DNNs im Profiling untersucht, sondern eine zufällige Auswahl an Schichten. Basierend auf den Messergebnissen bezüglich Latenz und Energiebedarf werden so die Metriken für alle anderen Layer abgeschätzt. Mit dem Framework PArtNNer [102] schlagen Ghosh et al. ein Verfahren ohne

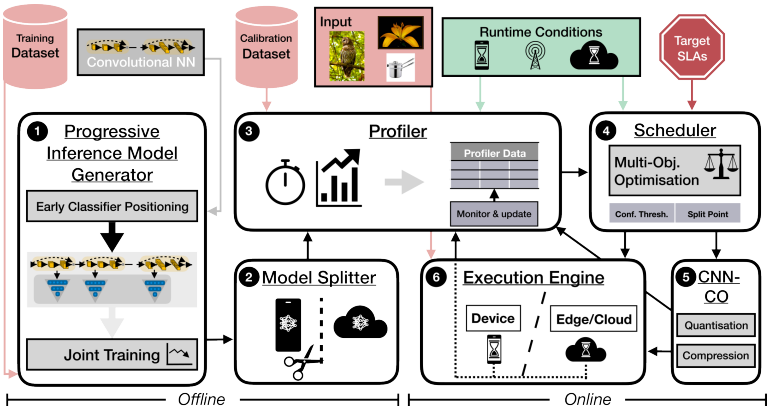


Abbildung 3.6: Übersicht des SPINN Frameworks aus [108]

offline Profiling vor. In diesem Fall werden zur Laufzeit Messungen durchgeführt und basierend auf den Ergebnissen nimmt der Algorithmus schließlich die Inferenz-Partitionierung vor.

Ähnlich der zuvor vorgestellten Ansätze fokussieren sich mehrere Verfahren [103, 104, 105] ausschließlich auf die Latenz des Systems und nutzen für die weitere Optimierung der Inferenz-Partitionierung zusätzlich Early Exits im DNN. Dabei handelt es sich um eine von Teerapittayanon et al. [106] vorgeschlagene Anpassung der DNN-Architektur. In ein bestehendes Netz werden dabei Exit Points eingeführt, an denen ein neuer Pfad hin zu einer Ausgabe angehängt ist. Am Ende dieses Pfads – bestehend aus verschiedenen Layern – ermittelt das Netzwerk einen Parameter, der angibt, mit welcher Wahrscheinlichkeit die Vorhersage des Netzes korrekt ist. Liegt diese über einem definierten Schwellwert, wird die Inferenz beendet und das Ergebnis ausgegeben. Andernfalls führt das System die nachfolgenden Layer des DNNs auf dem ursprünglichen Pfad aus. Im Vergleich zur Inter-Layer-Inferenz-Partitionierung unveränderter DNNs führt dies zu einer erhöhten Komplexität des Optimierungsproblems, da die Wahl eines frühen Exit Points signifikante Auswirkungen auf relevante Metriken hat [107]. Die Integration von Early Exits nutzt auch SPINN [108], welches dynamisch zur Laufzeit des Systems entscheidet, welcher Exit Point für die laufende Inferenz ausgewählt wird. Wie in Abbildung 3.6 dargestellt, ermittelt das Framework in der Offline-Phase zunächst Early Exits sowie alle potenziellen Partitionierungspunkte des trainierten DNNs. Zur Laufzeit ermittelt der Scheduler basierend auf den Runtime Conditions einen Partitionierungspunkt und gibt diese Information

an den Communication Optimizer (CO) zur Reduktion der Datenübertragung. Das Framework Boomerang [109] optimiert diese Wahl weiter mittels Deep Reinforcement Learning nach Wang et al. [110], indem es zusätzlich nach einem optimalen Partitionierungspunkt auf dem Early Exit Pfad sucht und somit eine Verringerung der Datenübertragung erzielt. Problematisch hierbei ist, dass die Verwendung von Early Exits abhängig von der DNN-Architektur einen Einfluss auf die Genauigkeit hat und damit auch ein entsprechendes Training der neu hinzugefügten Layer benötigt. Auswirkungen auf die Genauigkeit hat ebenfalls das von [111] vorgeschlagene Verfahren, welches in ihrem Framework Auto-Split Anwendung für Edge-Cloud-Systeme findet. Das Ziel besteht in der Reduzierung der Latenz durch Partitionierung und individueller Quantisierung der in der Edge-Plattform ausgeführten Layer unter Beachtung der Speicherbeschränkungen sowie der zulässigen minimalen Genauigkeit des DNNs.

Neben der Latenz bewerten einige Ansätze auch den Energiebedarf der vorhandenen Plattformen für verschiedene Partitionierungsschemata. Beispielsweise schlagen Ghasemi et al. [112] ein Framework vor, welches auf eine energieeffiziente Inferenz-Partitionierung abzielt. Dabei soll insbesondere ein geringer Energiebedarf im mobilen Endgerät für die Berechnungen und Datenübertragung erreicht werden. Sie formulieren das Problem in Form eines gewichteten Graphen und nutzen das Min-Cut-Verfahren, um einen optimierten Partitionierungspunkt zu ermitteln. Für die Gewichtung der Kanten ist dabei ein Profiling der Leistungsaufnahme des Geräts notwendig. Zusätzlich dazu bezieht das Framework JointDNN [113] die Limitierung durch den aktuellen Ladestand des Akkus mit ein. Je nach Anforderungen und Systemzustand wird somit dynamisch zwischen verschiedenen Partitionierungspunkten gewechselt, die entweder auf Performance oder Energieeffizienz optimiert sind.

Das Open-Source Framework DEFER [114] ermittelt die Partitionierungspunkte anhand der Anzahl von im System verfügbaren Plattformen und der zu erwartenden Latenz pro Layer und optimiert anschließend die Übertragung durch verlustlose Komprimierung der Daten. Ein ähnliches Verfahren nutzen Li et al. [115] in ihrem JALAD-Framework, welches auf Huffman-Codierung zur Verringerung der benötigten Bandbreite bei der Partitionierung der DNNs zurückgreift. Die Entwickler von DynO [116] eliminieren den aufwendigen Schritt der Codierung durch unabhängige Quantisierung der zu übertragenden Aktivierungen zwischen den Partitionen in Kombination mit Bit Shuffling. Dies ermöglicht nach Einschätzung der Autoren eine deutlich höhere Komprimierung im nachfolgenden Schritt. Neben klassischen Algorithmen zur Komprimierung existieren im aktuellen Stand der Technik auch Verfahren,

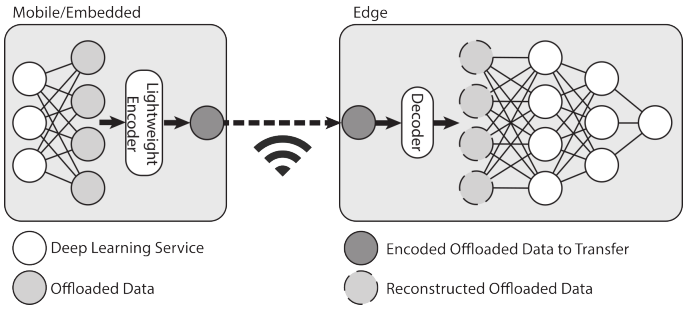


Abbildung 3.7: Übersicht des Deep Compressive Offloading Framework aus [118]

welche einen Autoencoder nach dem Partitionierungspunkt implementieren, wie beispielsweise in [117] vorgeschlagen. Das Deep Compressive Offloading Framework von Yao et al. [118] nutzt dies hauptsächliche zur Reduzierung der Übertragungslatenz am Partitionierungspunkt zwischen einem mobilen Endgerät und der Edge-Plattform. Wie in Abbildung 3.7 dargestellt, verringert das Verfahren durch Integration eines Encoders vor der Übertragung die Menge an Daten. Aufseiten der Edge-Plattform müssen die Daten entsprechend durch einen Decoder rekonstruiert werden, um die Inferenz fortzusetzen. In Zusammenspiel mit der Quantisierung der Aktivierungen und anschließender Codierung ermöglicht dieses Verfahren laut Autoren eine zwei- bis vierfache Reduktion der Gesamtlatenz bei einer Verschlechterung der Klassifizierungsgenauigkeit um 1 %. Eine ähnliche Methodik nutzen Hu et al. [119] für ihr Framework, welches eine hohe Auslastung beider Plattformen erzielen möchte, um den Datendurchsatz zu maximieren. Das Einfügen eines Autoencoders in ein DNN hat allerdings in der Regel Auswirkungen auf die Genauigkeit des DNNs. Dadurch ist ein zusätzliches Training für jeden potenziellen Partitionierungspunkt erforderlich, was eine schnelle Exploration des DNNs verhindert.

Alternativ hierzu stellen Eshratifar et al. [120] ein eigenes Layer vor, die Bottleneck Unit, welches im Training berücksichtigt wird. Als Beispiel nutzen die Autoren die Bildkompression mittels JPEG, welche nicht verlustfrei ist. Je nach Grad der Komprimierung wirkt sich dies entsprechend auf die Genauigkeit des Systems aus. Shao et al. [121] kombinieren die beiden zuvor genannten Techniken, indem sie zunächst die Aktivierungen mittels eines CONV- und Fully-Connected-Layers komprimieren und anschließend eine Codierung vornehmen. Dadurch erreichen sie eine deutliche Verringerung der benötigten Bandbreite für die Übertragung der Aktivierungen zwischen den Partitio-

nen und damit auch eine bessere Systemperformance. Dennoch ist auch in diesem Fall ein zusätzliches Training notwendig.

Die zuvor vorgestellten Verfahren befassen sich zumeist mit der Partitionierung zwischen Edge-Plattform und Cloud. Daneben ist eine Inferenz-Partitionierung aber auch innerhalb eines SoCs bestehend aus mehreren Recheneinheiten sinnvoll. Mit dieser Forschungsfrage befasst sich unter anderem das Framework AxoNN von Dagli et al. [122], welches nach einem optimierten Schedule für eine Nvidia Jetson Xavier AGX sucht. Diese Plattform besteht aus einer GPU und einem DLA, um die Inferenz von DNNs zu beschleunigen. In ihrer Arbeit betrachten sie dabei insbesondere die Kosten für die Übertragung zwischen den Einheiten über das Shared-Memory System. Die Partitionierung erfolgt anhand der durch Messungen abgeschätzten Latenz und dem Energiebedarf des Systems. Darüber hinaus präsentieren Van Delm et al. [123] eine Compiler-Toolchain namens HTVM, welche die Inferenz-Partitionierung vor der Generierung der Firmware durchführt und entsprechend die Hardwarecharakteristika des Systems berücksichtigt. Allerdings werden für das Mapping eines Layers auf einen Beschleuniger lediglich Hardwareparameter wie Bitbreite betrachtet, wodurch dieses Verfahren keine umfassende Exploration gewährleistet.

Sowohl die zuvor vorgestellten Methoden der Inter-Layer- als auch die der Intra-Layer-Inferenz-Partitionierung betrachten eine feste Hardwarekonfiguration und suchen für dieses System eine optimierte Aufteilung des gegebenen DNNs. Allerdings zeigen nachfolgende Publikationen, dass ein Hardware/-Software Co-Design zur Entwurfszeit basierend auf Simulationen vorteilhaft ist, um eine bessere Systemperformance zu erzielen. Unter anderem stellen LiKamWa et al. mit RedEye [124] einen analogen Beschleuniger vor, der direkt mit dem Pixel-Array einer Kamera verbunden ist. Durch die Anbindung dedizierter funktionaler Einheiten zur Ausführung von CONV- und Pooling-Layern, werden so bereits vor der Übermittlung der Daten an einen digitalen Prozessor die ersten Schichten eines DNN analog ausgeführt (s. Abbildung 3.8). Unter Zuhilfenahme einer entsprechenden Simulationsumgebung messen die Autoren eine signifikante Reduzierung des Energiebedarfs im System um 45 % im Vergleich zur reinen Ausführung im digitalen Prozessor. Die Resultate demonstrieren jedoch, dass analoge Beschleuniger, in Abhängigkeit von der Intensität des Rauschens, einen signifikanten Einfluss auf die Genauigkeit des DNNs ausüben.

Ko et al. [125] präsentieren in ihrer Arbeit ein Verfahren für ressourcenbeschränkte Application Specific Integrated Circuit (ASIC)-Plattformen. Dafür entwerfen sie basierend auf den Vorstudien zur Inferenz-Partitionierung eines

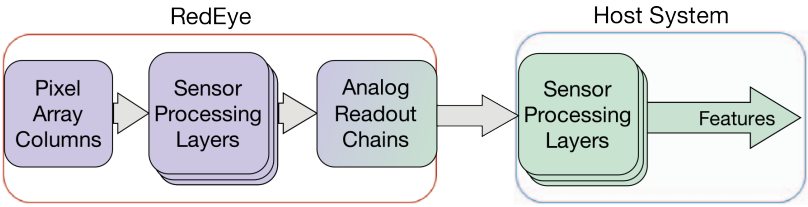


Abbildung 3.8: Übersicht der Inferenz-Partitionierung mit RedEye aus [124]

DNNs einen entsprechenden Hardwarebeschleuniger, welcher die Berechnungen mit gesteigerter Energieeffizienz durchführt. Zusätzlich verfügt die Plattform über einen JPEG-Encoder, um die zu übertragenden Zwischenergebnisse zu komprimieren. Auf diese Weise erzielt der Ansatz insgesamt eine deutlich höhere Energieeffizienz und einen höheren Durchsatz als bei der Inferenz ohne Partitionierung. Dennoch erfolgte die Aufteilung des Netzes hierbei nicht automatisiert, sodass sich aus den Ergebnissen zwar Richtlinien für andere Anwendungen ableiten lassen, eine direkte Übertragung aber nicht möglich ist.

Abschließend zu erwähnen ist CIM-fusion [126], welches ähnlich zu HybridAC [84] für die Inferenz-Partitionierung in heterogenen SoCs bestehend aus analogen und digitalen Beschleunigern verwendet werden kann. Im Gegensatz zu HybridAC nutzt dieses Verfahren die Inter-Layer-Inferenz-Partitionierung auf Basis des sogenannten Sensitivitätsfaktors. Dieser beschreibt, wie robust ein Layer gegenüber Rauschen ist und wird anhand von Messungen in existierenden PIM-Modulen ermittelt. Damit lässt sich der Einfluss auf die Genauigkeit des DNNs früh abschätzen und somit in die Aufteilung der Inferenz einbeziehen. Somit werden weniger robuste Layer auf dem digitalen Beschleuniger ausgeführt, während alle anderen die ungenauen analogen PIM-Beschleuniger des Systems nutzen. Dadurch erreicht die im Rahmen der Arbeit entwickelte Prozessorplattform eine hohe Leistungsfähigkeit bei geringen Genauigkeitseinbußen.

3.3 Ableitung der Anforderungen

Abschließend erfolgt in diesem Abschnitt eine zusammenfassende Analyse des aktuellen Stands der Technik. Tabelle 3.1 bietet eine übersichtliche Zusammenfassung zuvor vorgestellter und für die vorliegende Arbeit relevanter Frameworks unter Berücksichtigung verschiedener Gesichtspunkte.

Die Kategorisierung der verschiedenen Verfahren basiert dabei auf wesentlichen Eigenschaften, die für die Inferenz-Partitionierung von besonderer Bedeutung sind. Hierzu zählt das verwendete DNN-Framework, in dem die zu partitionierenden DNN-Modelle in das Framework gegeben werden müssen. Festzuhalten ist, dass die Methodiken hier unterschiedliche Tools verwenden, welche untereinander nicht kompatibel sind. Lediglich ein Verfahren, nämlich HTVM [123], existiert, welches Open Neural Network Exchange (ONNX) als Eingabe nutzt. Dies ermöglicht die Verwendung von Modellen aus verschiedenen DNN-Frameworks, da diese zumeist über eine Funktion zum Export der Netze im ONNX-Format verfügen.

Für die Partitionierung selbst sind neben der Unterscheidung zwischen Intra- und Inter-Layer-Inferenz-Partitionierung zwei weitere wesentliche Merkmale zu unterscheiden. Erstens erlauben viele Frameworks nicht mehrere Partitionierungspunkte (Mult. Part.) und unterstützen somit keine komplexeren Systeme bestehend aus mehreren Beschleunigerplattformen. Dies ist insbesondere im Falle der Inter-Layer-Inferenz-Partitionierung zu beobachten, da diese Verfahren häufig die Aufteilung zwischen einer Edge-Plattform und der Cloud betrachten bzw. diese zu optimieren versuchen. Dennoch existieren Frameworks wie AxoNN [122], die nachgewiesen haben, dass die Aufteilung eines DNNs über mehrere Partitionen von Vorteil ist. Der zweite wichtige Aspekt ist die Hardware-bezogene Partitionierung (HW-bezogene Part.), wobei für die Entscheidung der Partitionierung auch die Hardwarearchitektur und ihre Limitierungen einbezogen werden. Offensichtlich ist, dass dies meist nicht für Hardwareplattformen wie einem Raspberry Pi erfolgt, sondern in der Regel nur bei ASICs oder FPGAs als Zielplattform.

Die Unterstützung eines (Re-)Trainings ist lediglich bei einer geringen Anzahl von Frameworks integriert, da der Fokus in der Regel nicht auf funktionalen Optimierungsmetriken liegt. Die meisten Ansätze gehen davon aus, dass eine Quantisierung der Aktivierungen und Gewichte keinen nennenswerten Einfluss auf die Genauigkeit hat. Abgesehen von ODiMO [83], welches die Quantisierung der Gewichte aber nicht der Aktivierungen betrachtet, untersuchen lediglich Frameworks, welche das DNN durch hinzufügen von Layern manipulieren, die Auswirkungen auf dessen Genauigkeit. Generell ergeben sich für die Optimierungsmetriken eine Vielzahl an Kombinationen, wobei zumeist Latenz eine solche Metrik darstellt. Allerdings werden häufig nur zwei Kriterien für die Entscheidung der Inferenz-Partitionierung herangezogen. Lediglich Ko et al. [125] und HybridAC [84] betrachten mehrere Metriken, um eine optimierte Aufteilung des DNNs zu erzielen.

Zusammenfassend lässt sich daraus ableiten, an welcher Stelle es im aktuellen Stand der Technik an einem Framework zur Inferenz-Partitionierung fehlt. Anhand der Tabelle 3.1 ist klar zu erkennen, dass sich viele Forschungsgruppen mit der Aufteilung von DNNs zwischen Edge-Plattform und der Cloud bzw. innerhalb existierender Hardwareplattformen bestehend aus CPU und GPU bzw. DLA beschäftigen. In der Vergangenheit wurden hierfür bereits eine Vielzahl von Verfahren präsentiert, welche die Inferenz hinsichtlich verschiedener Metriken optimieren. Dagegen beschäftigen sich nur wenige wissenschaftliche Publikationen mit der Inferenz-Partitionierung als Teil des Hardware/Software Co-Designs zur Entwurfszeit des eingebetteten Systems. Hierzu zählen HybridAC [84] und ODiMO [83], welche die Intra-Layer Partitionierung bereits in einer frühen Entwicklungsphase untersuchen. Diese Verfahren eignen sich besonders für verteilte eingebettete Systeme im IoT, wo leistungsschwache Hardwareplattformen aufgrund stark beschränkter Ressourcen eingesetzt werden.

Die vorgestellten Verfahren demonstrieren aber auch, dass die Inter-Layer-Inferenz-Partitionierung sehr effektiv in hierarchisch aufgebauten Systemen sowie leistungsstarken Systemen funktioniert. In erstgenannter Umgebung ist eine parallele Verarbeitung von Teilschichten des DNNs nachteilig, da dies automatisch zu einer höheren benötigten Bandbreite zwischen den Plattformen führt. Ein Beispiel hierfür sind Systeme im Automobilbereich, welche hierarchisch von den Sensoren zur zentralen Recheneinheit aufgebaut sind. Des Weiteren ist zu berücksichtigen, dass eingebettete Plattformen, wie beispielsweise die Electronic Control Units (ECUs) in einem Fahrzeug, eine deutlich höhere Fläche sowie Leistungsaufnahme beanspruchen. Folglich werden Beschleuniger derart ausgelegt, dass sie gut auf die Layer der DNNs angepasst sind. Für die Inter-Layer-Inferenz-Partitionierung fehlt es derzeit allerdings an Methoden, welche eine solche Untersuchung für eingebettete Systeme im Automobilbereich oder Anwendungen mit ähnlichen Randbedingungen zur Entwurfszeit erlauben. Daraus lassen sich die in Tabelle 3.1 dargestellten Anforderungen an das im Rahmen der vorliegenden Arbeit entwickelte Framework ableiten.

	DNN Framework	Partitionierung	Multi. Part.	HW-bezogene Part.	(Re-) Training	Analyse-Verfahren	Ziel-Plattform	Optimierungsmetriken
Ko et al. [125]	-	Inter-Layer	✗	✓	✓	HW-Modell	Custom SoC	Energie, Durchsatz, Bandbreite, Genauigkeit
DeepThings [75]	Darknet	Intra-Layer	✓	✗	✗	Messung	CPU	Latenz, Durchsatz
Neurosurgeon [99]	Caffe	Inter-Layer	✗	✗	✗	Messung	CPU/GPU	Latenz, Energie
DADS [100]	Caffe	Inter-Layer	✗	✗	✗	Messung	CPU/GPU	Latenz, Durchsatz
Yao et al. [118]	TensorFlow	Inter-Layer	✗	✗	✓	Messung	CPU/GPU	Latenz, Bandbreite
BottleNet [120]	TensorRT	Inter-Layer	✗	✗	✓	Messung	CPU/GPU	Latenz, Energie
Shao et al. [121]	PyTorch	Inter-Layer	✗	✗	✗	Messung	CPU/GPU	Latenz, Bandbreite
JALAD [115]	-	Inter-Layer	✗	✗	✓	Messung	CPU/GPU	Latenz, Genauigkeit
SPINN [108]	PyTorch	Inter-Layer	✗	✗	✓	Messung	CPU/GPU	Latenz, Durchsatz, Genauigkeit
DynO [116]	PyTorch	Inter-Layer	✗	✗	✗	Messung	CPU/GPU	Latenz, Durchsatz, Genauigkeit
RoaD-RuNNer [101]	PyTorch	Inter-Layer	✓	✗	✗	Messung	CPU/GPU	Latenz, Energie
HTVM [123]	ONNX	Inter-Layer	✓	✓	✗	HW-Modell	ASIC	Latenz, Speicher
CIM-fusion [126]	-	Hybrid	✓	✓	✗	HW-Modell	Custom SoC	Energie, Genauigkeit
Elastic-DF [92]	PyTorch	Inter-Layer	✓	✓	✗	HW-Modell	FPGA	Latenz, Hardwareressourcen
M5 [93]	-	Inter-Layer	✓	✓	✗	Messung	FPGA	Durchsatz, Hardwareressourcen
AxoNN [122]	TensorFlow	Inter-Layer	✓	✓	✗	Messung	GPU/DLA	Latenz, Energie
HybridAC [84]	PyTorch	Intra-Layer	✓	✓	✗	HW-Modell	Custom SoC	Latenz, Energie, Durchsatz, Genauigkeit, Fläche
ODiMO [83]	PyTorch	Intra-Layer	✓	✓	(✓)	HW-Modell	Custom SoC	Latenz, Energie. Genauigkeit
Diese Arbeit	ONNX	Inter-Layer	✓	✓	✓*	HW-Modell	ASIC	Latenz, Energie, Durchsatz, Bandbreite, Genauigkeit, Fläche

*optional

Tabelle 3.1: Überblick über den aktuellen Stand der Inferenz-Partitionierung

Kapitel 4

Gesamtkonzept

Die vorliegende Arbeit beschreibt Ansätze der Inferenz-Partitionierung sowie damit einhergehende weitere Systemoptimierungen auf algorithmischer und technologischer Ebene. Das Gesamtkonzept ist dabei in den Prozess der Entwicklung von KI-Algorithmen und adäquater Hardwarearchitekturen eingebettet, um die Systemanforderungen möglichst früh in der Entwurfsphase einzubeziehen. Im Folgenden wird der allgemeine Entwicklungsprozess sowie die Integration der ressourcenbasierten Inferenz-Partitionierung in diesen Ablauf dargelegt. Des Weiteren erfolgt eine kurze Vorstellung der entwickelten Methoden für verschiedene Entwurfsebenen.

4.1 Rahmenbedingungen

Wie bereits in Abschnitt 2.1 erläutert, basieren viele aufkommende KI-Anwendungen auf DNNs. So finden sich beispielsweise im Bereich der Bildverarbeitung für Fahrassistenzsysteme der Zukunft häufig CNNs, da diese bessere Ergebnisse als klassische Verfahren liefern. Um eine hohe Genauigkeit der Algorithmen zu erzielen, muss im ersten Schritt nach der Sammlung von Daten zunächst eine NAS durchgeführt werden, sofern keine neuronalen Netze für eine ähnliche Anwendung vorliegen. Die NAS versucht dabei abhängig von der implementierten Methodik ein präzises DNN zu finden und nutzt dafür einen der Anwendung entsprechenden Testdatensatz [127]. Dieser Prozess nimmt jedoch bereits ohne Betrachtung des Gesamtsystems sehr viel Zeit und Hardwareressourcen in Anspruch. Somit ist die Einbeziehung der Systemanforderungen sowie der verfügbaren Hardware und damit auch der Inferenz-Partitionierung an dieser Stelle nicht oder nur auf Basis grober Abschätzungen möglich.

Eine effektives Hardware/Software Co-Design kann demnach erst erfolgen, sobald die exakten Hyperparameter des DNNs oder verschiedener Architektur-Kandidaten feststehen. Letzteres ist insbesondere deshalb sinnvoll, da die Quantisierung von Aktivierungen und Gewichten auf der Hardware unterschiedliche Auswirkungen auf die Genauigkeit eines DNNs hat. So liefert ein neuronales Netz zwar schlechtere Ergebnisse im Falle einer Inferenz mit Fließkommazahlen, dafür aber bessere Resultate im Vergleich zu anderen Architekturen nach der Quantisierung. Folglich lassen sich mehr als nur algorithmische Optimierungen an einem festen DNN während der Untersuchung der Inferenz-Partitionierung durchführen.

Ähnliches gilt für die Hardware des Gesamtsystems. Aufgrund der unterschiedlichen Anforderungen der einzelnen Layer des DNNs an den Beschleuniger ist es in der Regel schwer, eine perfekte Architektur hinsichtlich funktionaler und nicht-funktionaler Metriken für alle Schichten zu finden. Je nach Einsatzgebiet und vorhandenen Ressourcen ist es folglich vorteilhaft, mehr als einen Hardwarebeschleuniger einzusetzen. Dies hängt jedoch stark vom DNN ab und kann folglich nicht generisch festgelegt werden. Um eine optimierte Systemarchitektur zu ermitteln, ist in diesem Fall somit ein Hardware/Software Co-Design sinnvoll.

Das Gesamtkonzept der vorliegenden Arbeit ist in Abbildung 4.1 dargestellt. Es wird angenommen, dass die Suche nach einem neuronalen Netz mittels der NAS bereits abgeschlossen ist und ein Training dieser Architektur durchgeführt wurde. Demzufolge zielt die Methodik auf die Optimierung der Quantisierung auf der Hardware hinsichtlich Energieeffizienz und Performance ab. Hierfür notwendig sind dem DNN entsprechende Hardwarearchitekturen, welche die vorliegende Arbeit entweder aus dem aktuellen Stand der Technik entnimmt oder anhand eigener Konzepte auf algorithmischer und technologischer Ebene entwickelt. So kann eine optimierte Systemkonfiguration der Hardware für ein eingebettetes System aus den Untersuchungen abgeleitet werden. Zur finalen Realisierung ist daher lediglich die Nutzung eines ML Compilers notwendig, welcher einen ausführbaren Maschinencode für die verwendeten Hardwareplattformen generiert.

4.2 Entwurfsebenen

Die effiziente Ausführung der Inferenz neuronaler Netze erfordert Optimierungen auf verschiedenen Ebenen der Hardwareentwicklung. Dabei müssen

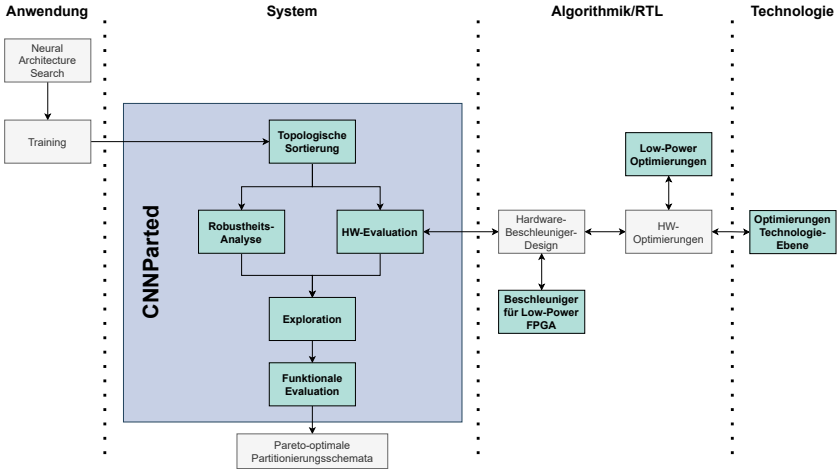


Abbildung 4.1: Übersicht des Gesamtkonzepts der vorliegenden Arbeit

je nach Anforderungen der Anwendung entsprechende funktionale und nichtfunktionale Metriken betrachtet und in die Suche nach einer bestmöglichen Systemarchitektur einbezogen werden. So sind im Bereich von HPC häufig Latenz und Datendurchsatz die dominierenden Optimierungskriterien, während IoT-Anwendungen in der Regel einen niedrigen Energiebedarf erfordern. Die vorliegende Arbeit basiert daher auf dem Entwurfsmodell für digitale Schaltungen gemäß Gajski et al. [128]. Demnach erfolgt eine Hardwareoptimierung in der Entwurfsphase auf verschiedenen Ebenen, um unterschiedlichen Einsatzgebieten und deren Anforderungen gerecht zu werden. Genauer optimiert das Gesamtkonzept ein System auf drei Ebenen, von System- bis Technologieebene, wie in Abbildung 4.1 aufgezeigt.

4.2.1 Systemebene

Im Zentrum der vorliegenden Arbeit steht die Optimierung der DNN-Inferenz für verteilte eingebettete Systeme mittels Partitionierung [Kre22a, Kre23a, Kre23b, Kre24a, Kre25]. Für das hierfür notwendige Hardware/Software Co-Design ist insbesondere die Betrachtung der Systemebene aus funktioneller und struktureller Sicht relevant. Dementsprechend betrachtet die entwickelte Methodik einerseits die Leistungsanforderungen und andererseits die Vernetzung der vorhandenen Hardwaremodule im System, um eine optimierte

Konfiguration hinsichtlich Hard- und Software zu ermitteln. Dabei erfolgt die Evaluation der Inferenz-Partitionierung auf abstrahierten Modellen realer Hardware, um die Simulationszeit signifikant zu reduzieren und somit eine umfangreichere Exploration des Entwurfsraums zu ermöglichen. So lassen sich basierend auf Abschätzungen mehrere Lösungsmöglichkeiten extrahieren, welche später unter realen Bedingungen auf ihre Anwendbarkeit hin getestet werden. Die Suche nach einer optimierten Verteilung der Arbeitslast im System umfasst nicht nur die Architektur von Hardwarebeschleunigern, sondern auch den Einfluss der verwendeten Links zwischen den Beschleunigerplattformen. Zudem evaluiert die Methodik auch die für die Ausführung der DNNs auf der Hardware erforderliche Quantisierung von Aktivierungen und Parametern in Bezug auf die Auswirkungen auf die Genauigkeit des Netzes. Im Wesentlichen nutzt das Verfahren folglich ein trainiertes DNN, welches im Rahmen einer NAS für eine bestimmte Aufgabe entwickelt wurde, und untersucht dieses auf unterschiedliche Partitionierungsschemata für eine mögliche Hardwarekonfiguration hinsichtlich relevanter funktionaler und nicht-funktionaler Metriken (z.B. Genauigkeit und Latenz). Auf dieser Grundlage lässt sich abschätzen, ob weitere Optimierungen auf tieferen Entwurfsebenen für die Einhaltung der anwendungsspezifischen Systemanforderungen erforderlich sind.

4.2.2 Algorithmische Ebene

Insbesondere im Kontext von IoT sind häufig zusätzliche Optimierungen notwendig, um die Ausführung der Inferenz in solchen Geräten zu realisieren. Wenig optimierte Systeme können zum Teil weder die Anforderungen an die Latenz bzw. den Datendurchsatz noch die an den maximalen Energiebedarf erfüllen. Dabei kommt neben den Hardwarebeschleunigern auch der verwendeten drahtlosen Kommunikationsschnittstelle, wie z.B. Bluetooth Low Energy (BLE), eine besonders wichtige Bedeutung zu, da diese einen großen Einfluss auf die genannten Metriken hat. Die vorliegende Arbeit führt folglich Optimierungen auf algorithmischer Ebene durch. Einerseits werden softwareseitig Methoden zur Reduktion der benötigten Bandbreite angewandt [Kre22b], um den Energiebedarf der Schnittstelle zu reduzieren und gleichzeitig den Datendurchsatz zu erhöhen. Zum Anderen demonstriert die vorliegende Arbeit, welche Optimierungsstrategien für verschiedene Hardwareplattformen einschließlich FPGAs zur Erfüllung der Systemanforderungen führen [Kre23c, Kre24c]. Dies umfasst die Konzeption von Beschleunigern, welche den zu erwartenden Datenfluss möglichst effizient in Hardware

abbilden, sowie die Nutzung von Approximate Computing zur Reduzierung der Rechenkomplexität.

4.2.3 Technologieebene

Zuletzt stellt die vorliegende Arbeit eine Optimierungsstrategie auf Technologie-, genauer Gatter- und elektrischer Ebene für Anwendungen vor, welche speziellen Rahmenbedingungen unterliegen. Bei sogenannten Energy-Harvesting-Systemen erfolgt die Gewinnung der benötigten elektrischen Energie aus der Umgebung, beispielsweise mittels Solarzellen. Derartige Systeme finden häufig Anwendung, um Daten in schwer zugänglichen Gebieten zu erfassen. Folglich ist ein einfaches Austauschen von Batterien oder das Zurücksetzen der Hard- und Software vor Ort kaum möglich. Solche sensornahen Geräte müssen daher abgesichert werden, um eine Verarbeitung der Daten über einen langen Zeitraum zu ermöglichen. Dieser Aspekt ist somit auch relevant für die Inferenz-Partitionierung in verteilten eingebetteten Systemen. Um eine entsprechende Ausfallsicherheit zu gewährleisten bzw. sicherzustellen, dass relevante Messdaten ausgewertet werden, sind Optimierungen auf Technologieebene erforderlich [Kre23d]. Dabei sorgen nichtflüchtige Speicherzellen für die Sicherung des aktuellen Systemzustands auch bei einem Ausfall der Spannungsversorgung. Neben Energy-Harvesting-Systemen kann dieser Ansatz auch in sicherheitskritischen Anwendungen wie Fahrerassistenzsystemen eingesetzt werden, damit das Fahrzeug im Falle einer Störung schnell in einen sicheren Zustand übergeht und nicht komplett neu starten muss.

Kapitel 5

Inferenz-Partitionierung von DNNs

Die Zahl der Sensoren in Anwendungen wie dem autonomen Fahren nimmt ständig zu. Daraus lässt sich ableiten, dass eine Verarbeitung der Daten auf einer zentralen Plattform aufgrund der damit einhergehenden Anforderungen hinsichtlich der Bandbreite und Performance nur schwer realisierbar ist. Aus diesem Grund werden vermehrt Ansätze gesucht, welche dieses Problem lösen. Ein solcher ist die Inferenz-Partitionierung, wodurch eine bessere Aufteilung der Arbeitslast und je nach DNN auch eine geringe Übertragungsbandbreite zwischen den Plattformen im verteilten System erzielt wird. In diesem Kapitel erfolgt zunächst eine Untersuchung des Potenzials der Inferenz-Partitionierung für ausgewählte Systeme und Anwendungen im Rahmen zweier Vorstudien. Im Anschluss wird eine mathematische Problemdefinition formuliert und ein generisches Framework namens CNNParted präsentiert, welches die Exploration verschiedener Partitionierungsschemata ermöglicht. Abschließend folgt eine Evaluierung des Frameworks anhand zweier gängiger Anwendungsfälle.

5.1 Vorstudie I: Partitionierung in ASIC-basierten Systemen

Aktuelle eingebettete Systeme bestehen häufig aus verteilten Rechenplattformen, die eine lokale Vorverarbeitung der Daten am Sensor ermöglichen. Diese sind in der Regel als ASIC implementiert, um eine möglichst hohe Energieeffizienz und einen geringen Flächenverbrauch zu erzielen. Mittels Vorverarbeitung der Daten tragen diese Plattformen zu einer Reduktion der zu übertragenden Datenmenge an die zentrale Steuereinheit bei. In vielen Kamera-basierten eingebetteten Anwendungen, z.B. in der Automobil- oder Robotertechnik, reicht die Vorverarbeitung der Daten in den einzelnen Knoten

jedoch nicht immer aus, um eine Überlastung des Bordnetzes zu vermeiden. Solche Systeme sind folglich durch eine maximale Anzahl möglicher Sensoren im gesamten System eingeschränkt.

Dennoch ist in den vergangenen Jahren eine steigende Anzahl an Sensoren in solchen Systemen zu beobachten. Indem Teile der Arbeitslast der zentralen Edge-Plattform bereits im Sensorknoten verarbeitet werden, kann ebenjener Bedarf an zusätzlichen Sensoren für den Einsatz in z.B. Assistenzrobotern oder Fahrerassistenzsystemen gedeckt werden. Allerdings sind eingebettete Systeme in Bezug auf Energiebedarf und Latenzzeit eingeschränkt, da sie mit einem begrenzten Energiebudget und Echtzeitanforderungen arbeiten. Ziel dieser ersten Vorstudie ist daher zu untersuchen, welchen Einfluss eine Verteilung der Arbeitslast zwischen einem hoch spezialisierten Hardwarebeschleuniger am Sensor und einer HPC-Plattform in der Edge-Einheit auf Energiebedarf und Latenz in ASIC-basierten System hat. Die Ergebnisse der nun folgenden Vorstudie sowie die Erläuterungen wurden durch den Autor in [Kre22a] veröffentlicht.

5.1.1 Übersicht

Das partielle Auslagern der CNN Inferenz auf andere Plattformen im System erhöht dabei die Komplexität im Hardware/Software Co-Design, da zusätzliche Randbedingungen betrachtet werden müssen. Aus diesem Grund wurde im Rahmen der Vorstudie eine Simulations-Toolchain entwickelt, welche eine schnelle Exploration des Entwurfsraums ermöglicht. Ziel ist es, anhand der Simulationsergebnisse Aussagen über mögliche Hardwareoptimierungen in der sensornahen Plattform zu erlauben und die Effizienz der Inferenz-Partitionierung in ASIC-basierten Systemen nachzuweisen. Dazu betrachtet die Toolchain zur Reduktion der Komplexität ein festes System bestehend aus einer sensornahen Rechenplattform und einer zentralen Edge-Einheit, welche über ein Bussystem miteinander verbunden sind. Eine Übersicht über deren Aufbau ist in Abbildung 5.1 gegeben.

Die Simulationsumgebung basiert im Wesentlichen auf in der Wissenschaft häufig verwendeten Software-Tools, um die Kompatibilität zu Open-Source Deep Learning Frameworks wie PyTorch [129] zu gewährleisten. Als Eingabe benötigt die Toolchain folgende Informationen:

- Die Beschreibung der Hardwarearchitektur in der sensornahen Plattform, welche den verwendeten KI-Beschleuniger sowie die Speicherhierarchie umfasst.

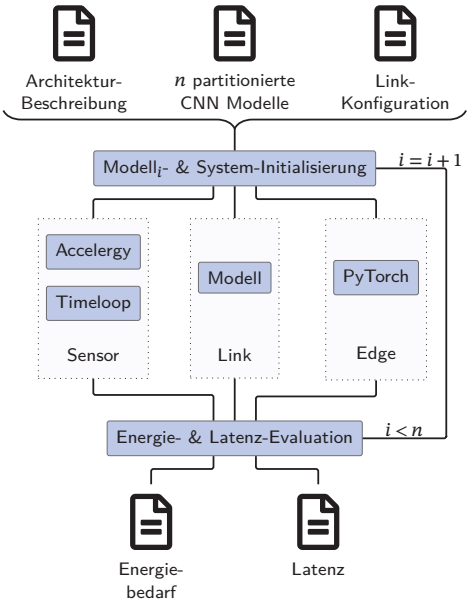


Abbildung 5.1: Übersicht der Simulations-Toolchain für die erste Vorstudie zur Abschätzung von Energiebedarf und Latenz

- Eine Menge an zuvor manuell partitionierten CNN Modellen abgeleitet aus einem CNN, welche nacheinander untersucht werden sollen.

- Die Konfiguration des Bussystems zur Abschätzung des Energiebedarfs und der Latenz.

Um die Gesamtsimulationszeit für die Evaluationen deutlich zu verkürzen, ist die Toolchain nach der System-Initialisierung in drei parallele Segmente unterteilt. Diese simulieren oder messen dabei den Energiebedarf und die Latenz eines gegebenen CNN für das jeweilige Systemmodul, d.h. für die sensornahe Plattform, das Bussystem und die Edge-Plattform. In der anschließenden Auswertung werden die Ergebnisse gesammelt, addiert und in einer Tabelle gespeichert, bevor im Anschluss das nächste partitionierte CNN evaluiert wird. Nachdem alle CNN Modelle der gegebenen Menge untersucht wurden, gibt die Toolchain den Gesamtenergiebedarf und die Inferenz-Latenz für jedes neuronale Netz zur weiteren Analyse durch den Nutzer aus.

5.1.2 Systemmodellierung

Für eine erste Abschätzung der Auswirkungen der Inferenz-Partitionierung auf den Gesamtenergiebedarf und die Latenz von verschiedenen CNNs ist keine exakte Bestimmung der Metriken notwendig, da dies das physische Vorhandensein entsprechender Hardware erfordert. Dennoch sind hinreichend gute Modelle der Systemkomponenten erforderlich, um allgemeine Aussagen anhand der Evaluationsergebnisse abzuleiten.

Sensornahe Plattform

Die Hardware-orientierte Auswertung von CNNs ist besonders im Bereich der eingebetteten Systeme wichtig, da diese in der Regel sowohl durch den maximalen Energiebedarf als auch durch die maximale Latenz eingeschränkt sind. Aus diesem Grund werden in solchen Plattformen häufig hoch spezialisierte Hardwarebeschleuniger integriert, um die Inferenz effizient und hochperformant auszuführen. Es existieren einige Hardwarearchitekturen [40, 41, 47, 130], von denen jede ihre eigenen Vor- und Nachteile in Bezug auf Performance, sowie Energie- und Flächenverbrauch hat. Das optimale Design für eine gegebene Anwendung zu finden ist daher keine triviale Aufgabe. Um eine schnelle Simulation von CNN-Beschleunigern in diesen ASICs und damit einhergehend die Bewertung ebenjener für eine gegebene Anwendung zu ermöglichen, ist daher ein schnelles und flexibles Framework erforderlich. Software-Tools wie beispielsweise FLECSim [Hot21] und andere [131, 132] ermöglichen eine Zyklen-basierte Simulation und Bewertung einer breiten Auswahl von Beschleunigerarchitekturen. Die Analyse jedes einzelnen Taktzyklus führt allerdings zu einer hohen Simulationszeit und ist wie bereits erwähnt nicht erforderlich, um die Auswirkungen der Inferenz-Partitionierung auf die Systemperformance abzuschätzen. Im Gegensatz dazu liefern Simulationsumgebungen auf Systemebene zur Evaluierung von CNN-Beschleunigern, wie z.B. die Frameworks in [133, 134, 135, 136, 137, 138], zwar quantitativ weniger präzise Ergebnisse, dafür aber eine deutlich reduzierte Simulationszeit. Es sei zudem darauf verwiesen, dass die Mehrzahl dieser Tools an eine spezifische Hardwarearchitektur, z.B. systolische Arrays, gebunden ist und lediglich die Anpassung weniger Architekturparameter erlaubt. Des Weiteren wird die Optimierung von Verarbeitungselementen oder der Speicherhierarchie in den meisten Fällen nicht berücksichtigt.

Um eine möglichst hohe Flexibilität in Bezug auf die CNN-Beschleuniger zu bieten, wurde für diese Vorstudie das Tool Timeloop [139] in die Toolchain

integriert. Es nutzt ein analytisches Modell des Hardwarebeschleunigers als Eingabe, sucht mittels eines Mappers nach einer optimalen Abbildung der Inferenz einer einzelnen CNN-Schicht auf die Architektur und schätzt abschließend den Energiebedarf sowie die Latenz pro Layer ab. Dabei bietet Timeloop verschiedene Suchalgorithmen und Optimierungsmetriken, z.B. Energie und Latenz, sowie einstellbare Bedingungen für die Beendigung der Exploration. Diese Methodik ermöglicht somit eine faire Bewertung verschiedener Beschleunigerarchitekturen und DNN-Anwendungen in einer angemessenen Simulationszeit.

Wie von Guo et al. [140] nachgewiesen, tragen CONV in der Regel mit Abstand am meisten zur Ausführungszeit von CNNs bei. Aus diesem Grund fokussiert sich die Evaluation der sensornahen, eingebetteten Plattform auf die Abschätzung des Energiebedarfs und der Latenz ebenjener CONVs. Durch diesen Ansatz lässt sich die Simulationszeit weiter reduzieren, was insbesondere im Falle größerer CNNs sinnvoll ist.

Für die Analyse der Layer benötigt Timeloop neben einer Beschreibung der Hardwarearchitektur die Konfiguration des internen Mappers sowie die Größe des Eingangstensors. Letztere unterscheidet sich dabei zwischen den verschiedenen CONVs in einem CNN. Aus diesem Grund muss die Simulationsumgebung zunächst die Eingangsgrößen der Schichten aus dem gesamten Netzwerk extrahieren, bevor eine Evaluation mittels Timeloop erfolgen kann. Basierend auf diesen Informationen wird anschließend die Anzahl der benötigten Taktzyklen des Hardwarebeschleunigers für die Berechnung einer CONV Inferenz durch das Tool ermittelt.

Die Berechnung des Energiebedarfs wurde in dieser Vorstudie mit dem Open-Source-Tool Accelergy [141] durchgeführt, welches die Metrik anhand von Action Counts jedes einzelnen Moduls berechnet. Action Counts sind dabei Zähler, welche bei jedem Lese- oder Schreibzugriff auf ein Register erhöht werden. Die Werte ebenjener Zähler werden dabei durch Timeloop ermittelt, nachdem ein passendes Mapping des CONVs auf den Beschleuniger gefunden wurde. Accelergy nutzt sogenannte Primitive Component Libraries zur Abschätzung des Gesamtenergiebedarfs der verwendeten Hardware. Dabei handelt es sich um Lookup-Tabellen, welche für verschiedene Technologieknoten, z.B. für 65 nm oder 28 nm, den Energiebedarf einzelner Lese- oder Schreibvorgänge einer bestimmten Komponente bereithalten. Hierfür macht sich Accelergy verschiedene Plug-Ins zunutze. So wurde für die Modellierung der Speicher das Tool CACTI [142] verwendet, während die Abschätzung des Energiebedarfs der logischen Komponenten im Beschleuniger auf Aladdin-basierten Tabellen [143] aufbaut. Auf diese Weise lässt sich der Energiebedarf

des Beschleunigers mittels Accelergy in Kombination mit Timeloop realitätsnah abbilden.

Bei Aladdin handelt es sich um ein Framework zur Modellierung von Hardwarebeschleunigern vor der Implementierung auf Register Transfer Level (RTL)-Ebene [143]. Für die Abschätzung von Energiebedarf und Performance nutzt es einen ressourcenbeschränkten, dynamischen Datenabhängigkeitsgraph und führt mehrere Optimierungsschritte durch, wie beispielsweise Loop Rolling und Pipelining. Allerdings ist das Framework nicht in der Lage, Ressourcen außerhalb des Beschleunigers zu modellieren, weshalb hierfür eine weitere Modellierungsumgebung benötigt wird. CACTI ist ein Tool zur Modellierung von on-chip und off-chip Caches und Speicher hinsichtlich Zugriffszeit, Leistungsaufnahme und Flächenverbrauch. Es wurde erstmals 1996 vorgestellt [144] und seither kontinuierlich entsprechend der Entwicklung im Bereich der Speichertechnologien erweitert und optimiert [145]. Dadurch findet es auch heute noch Anwendung und wird als Basis für Weiterentwicklungen wie beispielsweise der Modellierung von FinFET Caches und Speicher genutzt [146]. In Kombination mit Aladdin lassen sich so die relevanten Systemmetriken sinnvoll abschätzen und darauf basierend Rückschlüsse auf die Effektivität der Inferenz-Partitionierung ziehen.

Ethernet Modell

Neben der Modellierung des Beschleunigers in der sensornahen Plattform kann auch die Verbindung zur Edge-Plattform eine wichtige Rolle in Bezug auf Gesamtlatenz und -Energiebedarf einnehmen. Wichtiger jedoch ist die Berücksichtigung der verfügbaren Bandbreite, die verhindern kann, dass die Partitionierung an jedem beliebigen Punkt innerhalb des CNN durchführbar ist. In dieser Vorstudie wurde eine kabelgebundene Ethernet-Verbindung betrachtet, wie diese in [Kre22a] beschrieben ist.

Für die Modellierung wurde die Annahme getroffen, dass der eingebettete Mikroprozessor direkt mit der Physical layer (PHY) Komponente verbunden ist. In diesem Kontext ist zu erwähnen, dass die Leistungsaufnahme des PHY jene des Links zwischen sensornaher Plattform und der zentralen Edge-Einheit dominiert. In der Konsequenz fokussiert sich die Modellierung des Übertragungsprotokolls auf dieses Systemmodul.

Ethernet ist in verschiedenen Varianten verfügbar, die sich dabei insbesondere in der maximal möglichen Übertragungsrate unterscheiden. Dies hat allerdings auch einen Einfluss auf den Energiebedarf des PHY, weshalb für

jeden Typ andere Werte angenommen werden müssen. Das in [Kre22a] vorgestellte Ethernet Modell basiert auf typischen Leistungsaufnahmen, welche in [147] beschrieben sind.

Darüber hinaus hat auch die Übertragungsdauer T_{link} der Daten einen gewichtigen Einfluss auf die Gesamtlatenz der Inferenz und muss daher evaluiert werden. Im Falle von Ethernet hängt diese im Wesentlichen von der Datenmenge N_{data} sowie der Übertragungsrate B ab. Basierend auf der Arbeit von Gámiz-Caro et al. [148], ist die Latenz für die Übertragung eines Datenpakets via Ethernet dabei gegeben durch

$$T_{link} = \frac{N_{hdr} + N_{stf} + max\{N_{data}, N_{min}\}}{B} + T_{wire} \tag{5.1}$$

mit den Konstanten $N_{hdr} = 20$, $N_{stf} = 18$ und $N_{min} = 46$ für die Größe des Headers, der umschließenden Segmente zur Separation der Pakete bzw. der minimalen Anzahl an Nutzdaten in Bytes. Die Laufzeitverzögerung T_{wire} ist dagegen abhängig von der Kabellänge und der Übertragungsgeschwindigkeit des verwendeten Materials. Für Kupfer wird eine Ausbreitungsgeschwindigkeit von $6 \cdot 10^{-9}\,\mathrm{s\,m^{-1}}$ angenommen, woraus für eine bestimmte Kabellänge l_{line} in Metern folgt:

$$T_{wire} = l_{line} \cdot 6 \cdot 10^{-9}\,s \tag{5.2}$$

Edge-Plattform

Im Gegensatz zur sensornahen Plattform und dem Ethernet Link wurde die Edge-Einheit für diese Vorstudie nicht modelliert, sondern angenommen, dass eine frei auf dem Markt erhältliche Plattform im System zum Einsatz kommt. Deren Architektur ist häufig nicht als Modell verfügbar, somit muss die Performance durch Messungen ermittelt werden. Dieses Szenario ist nicht unüblich, da man als Entwickler nicht immer die Möglichkeit hat, Einfluss auf die verwendete Hardware auf beiden Seiten des Links zu nehmen. Somit kann abhängig von den Systemanforderungen und den Anwendungen eine High-Performance Plattform wie die NVIDIA Jetson TX-2 oder eine eingebettete CPU Plattform wie beispielsweise ein Raspberry Pi für die Evaluation der Inferenz-Partitionierung verwendet werden.

Für die Evaluation der Latenz des CNNs nutzt die verwendete Toolchain das PyTorch Benchmark Modul [129], welches genau für solche Messreihen

entwickelt wurde und auf vielen Plattformen verfügbar ist. Es erlaubt eine Parallelisierung der Anwendung durch die Nutzung mehrerer Threads und erzielt durch ein vorangestelltes Warm-Up sowie durch Synchronisation der CUDA Kerne mit der Host-CPU zuverlässigere Messergebnisse. Insgesamt werden zur Ermittlung der Latenz 1000 Messungen durchgeführt und der Median als Ergebnis ausgegeben. Durch die Verwendung des Median anstelle des Mittelwerts ist eine erhöhte Robustheit gegenüber Ausreißern sichergestellt, was die Qualität der Messreihe zusätzlich verbessert.

Die Evaluation der Leistungsaufnahme solcher Plattformen ist dagegen schwierig, da diese häufig über keine Messpunkte auf den Platinen verfügen. Darüber hinaus würde die Messung durch parallel ausgeführte Anwendungen und das Betriebssystem selbst gestört werden. Da die Vorstudie besonders auf eine Untersuchung der Sensor-Plattform abzielt und demnach der Energiebedarf der Edge-Plattform von geringerem Mehrwert ist, wurde dieser daher als konstant angenommen.

5.1.3 Testaufbau

Die verwendete Simulation zur Ermittlung der Metriken für die sensornahe Plattform und den Link wurde auf einem AMD EPCY 7702P Prozessor mit CentOS ausgeführt. Für die Evaluation eines Latenz- und Energie-optimalen Mappings der Layer auf den verwendeten Hardwarebeschleuniger werden Timeloop acht parallele Threads zur Verfügung gestellt. Timeloop wurde derart konfiguriert, dass es eine optimierte Form der linearen Suche verwendet, welche sich Pruning-Methoden zunutze macht. Damit werden irrelevante Permutationen des Datenflusses durch die Hardwarearchitektur noch bevor deren Evaluation gefiltert, wodurch die Suche beschleunigt wird. Als Abbruchkriterium wurden 100 valide aber suboptimale, aufeinanderfolgende Mappings festgelegt, was einen guten Kompromiss zwischen Qualität der Ergebnisse und der Simulationszeit darstellt.

Im Rahmen der Vorstudie wurde ein NVIDIA Jetson TX-2 als eingebettete Edge-Plattform verwendet. Die darin integrierte GPU verfügt über 256 Pascal-Kerne mit einer Taktfrequenz von 1,3 GHz, welche für die Verarbeitung der ML-Aufgaben zuständig sind [149]. In Kombination mit der Quadcore-ARM-CPU weist diese Plattform insgesamt eine hohe Energieeffizienz auf, da sie auf der höchsten Leistungsstufe, der MAX-P-Konfiguration, lediglich 15 W aufnimmt. Die Durchführung der Messreihen in der Edge-Einheit erfolgte unter Verwendung von PyTorch in Kombination mit den

entsprechenden CUDA-Bibliotheken auf einem unveränderten 64-Bit-Jetson-Linux-Betriebssystem.

Anwendungen

Beispielhaft werden in dieser Vorstudie drei verschiedene CNN Topologien hinsichtlich ihrer Eignung für die Inferenz-Partitionierung auf einer ASIC-Plattform untersucht, welche auch in der PyTorch-Bibliothek Torchvision verfügbar sind:

- Fully Convolutional Network [150] mit ResNet-50 Backbone (*FCN-ResNet50*), welches zur semantischen Segmentierung beispielsweise im Bereich der Fahrerassistenzsysteme eingesetzt wird,
- *GoogLeNet* [151], welches häufig für die Klassifizierung von Bilder genutzt wird und
- *SqueezeNet V1.1* [152], welches ebenfalls Anwendung in der Klassifizierung von Bildern angewandt wird.

Die Inferenz-Partitionierung von CNNs ist nicht in jedem Fall sinnvoll. Aktuelle Netze nutzen sehr häufig sogenannte *Skip Connections*, um das Vanishing Gradient Problem im Training abzuschwächen. Dabei werden aufgrund der Tiefe des Netzes ab einem bestimmten Punkt bei der Fehlerrückführung und Aktualisierung der Gewichte im Trainingsprozess keine spürbaren Veränderungen mehr am Netz vorgenommen, da der angewandte Gradient mit jedem Layer schrumpft. Es findet folglich kein Training der ersten Layer des Netzes statt, die veränderbaren Parameter bleiben konstant. Die Skip Connection ist in der Regel Teil eines Block-Moduls im CNN, welches aus mehreren parallelen Pfaden besteht. Eine Partitionierung innerhalb eines solchen Blocks ist daher häufig unvorteilhaft, weil dadurch mehr Daten übertragen werden müssen und auch mehr Speicherplatz notwendig ist. Aus diesem Grund gibt es abhängig vom gegebenen CNN gewisse Punkte, an denen eine Partitionierung weniger sinnvoll ist und daher bei der Evaluation nicht berücksichtigt werden muss. Für ein tieferes Verständnis der gewählten Partitionierungspunkte, soll dies nachfolgend jeweils für die drei ausgewählten Testanwendungen näher erläutert werden. Ein Partitionierungspunkt entspricht dabei dem letzten ausgeführten CNN-Layer auf der sensornahen Plattform, also dem Layer vor der Übertragung der Zwischenergebnisse an die Edge-Plattform.

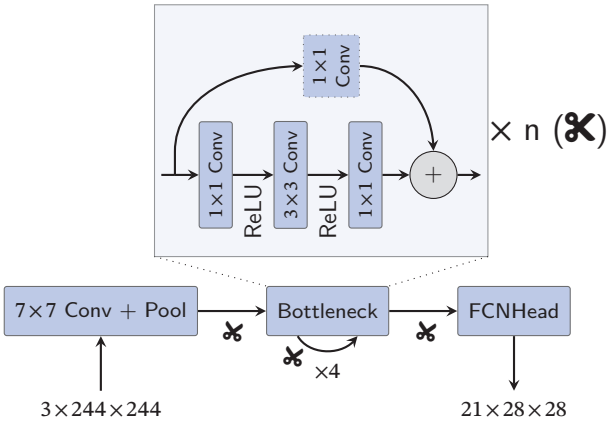


Abbildung 5.2: Übersicht der FCN-ResNet Architektur mit den potenziellen Partitionierungspunkten dargestellt durch die Scheren-Symbole

Abbildung 5.2 zeigt die detaillierte Struktur eines FCN-ResNet sowie die Stellen an denen eine Partitionierung des Netzes sinnvoll sein kann. Das FCN-ResNet wird häufig mit ResNet-50 oder ResNet-101 als Backbone verwendet, da dieses hiermit recht hohe Genauigkeiten erzielt. Es besteht aus zwei konfigurationsunabhängigen Teilen an Anfang und Ende, FCN-Head und FCN-Tail, sowie vier *Bottleneck* Blöcken in der Mitte. Jeder dieser Blöcke verfügt über n residuale Blöcke, wobei eine Partitionierung innerhalb dieser aufgrund der zwei parallelen Pfade (s. Abbildung 5.2 oben) nicht sinnvoll ist. Die Anzahl der residualen Blöcke in einem *Bottleneck* ist durch die Konfiguration des Netzes, d.h. dem ResNet-Backbone definiert. So verfügt beispielsweise ein ResNet-50 in den jeweiligen *Bottleneck* Modulen über 3, 4, 6 bzw. 4 residuale Blöcke. Das FCN-Head besteht aus einem großen CONV sowie einem Max Pool Layer zur Reduktion der Feature Map. Dagegen nutzt das FCN-Tail zwei CONV Layer für die segmentierte Ausgabe des Netzes, wobei das hier in der Vorstudie verwendete CNN für 21 Klassen trainiert wurde. Im Falle von FCN-Head und FCN-Tail kann eine Partitionierung zwischen jedem einzelnen Layer erfolgen, dies ergibt insgesamt 25 potenzielle Partitionierungspunkte für FCN-ResNet50.

Für GoogLeNet und SqueezeNet V1.1 wird die Aufteilung zwischen den Plattformen auf ähnliche Weise durchgeführt. Erst genanntes besteht aus mehreren sogenannten *Inception* Modulen, welche über verschiedene Pfade mit unterschiedlichen Filtergrößen der CONVs zwischen den Layer verfügt. Eine

Partitionierung innerhalb dieser Module ist daher nicht vorteilhaft. Daneben verfügt GoogLeNet über sequenzielle CONVs sowie Pooling und Fully-Connected Layers, zwischen denen eine Aufteilung des Netzes möglich ist. So ergeben sich für dieses CNN entsprechend insgesamt 19 potenzielle Partitionierungspunkte.

Die Architektur von SqueezeNet is darauf optimiert, möglichst wenige Parameter zu nutzen, um die Speicherauslastung zu reduzieren. Ähnlich den *Inception* Modulen von GoogLeNet, verfügt das Netz über sogenannte *Fire* Blöcke, welche eine Feature Extraction mittels zwei parallelen CONVs durchführt. In SqueezeNet sind diese Blöcke ebenfalls hintereinander angeordnet, gefolgt von einem Fully-Connected Layer für die Klassifizierung am Ende. Insgesamt ergeben sich dadurch für diese erste Vorstudie 17 zu untersuchende Partitionierungspunkte.

5.1.4 Evaluation

Für die Auswertung der Auswirkungen durch Inferenz-Partitionierung der genannten CNNs wurde in dieser Vorstudie ein eingebettetes System angenommen, welches typischerweise im Automobil oder auch in der Robotik zu finden ist. Neben der zentralen GPU verfügt dieses im sensornahen Knoten über einen hochspezialisierten Hardwarebeschleuniger in einer Low-Power Domäne. Die Nutzung einer solchen erlaubt eine einfachere und leichtgewichtigere Verkabelung, was in den genannten Einsatzgebieten von Vorteil ist. Beide Plattformen sind über ein fünf Meter langes Gigabit-Ethernet Kabel miteinander verbunden. Für FCN-ResNet50 sowie GoogLeNet kommt aufgrund der Größe der beiden CNNs ein Hardwarebeschleuniger ähnlich der Simba-Architektur [40] zum Einsatz, welcher mit 500 MHz getaktet ist. Dagegen wurde für SqueezeNet V1.1 ein kleinerer Beschleuniger mit geringerem Energiebedarf für das System angenommen. Angelehnt an die Eyeriss-Architektur [47], wurde dieser mit einer Taktfrequenz von 200 MHz simuliert. Aufgrund der Annahme, dass die Leistungsaufnahme der Edge-Plattform konstant ist, werden nachfolgend nur der Energiebedarf der sensornahen Plattform sowie des Links betrachtet.

FCN-ResNet50

Die Ergebnisse für die Abschätzung der Gesamtlatenz sowie des Gesamtenergiebedarfs pro Partitionierungspunkt sind in Abbildung 5.3 dargestellt. Im

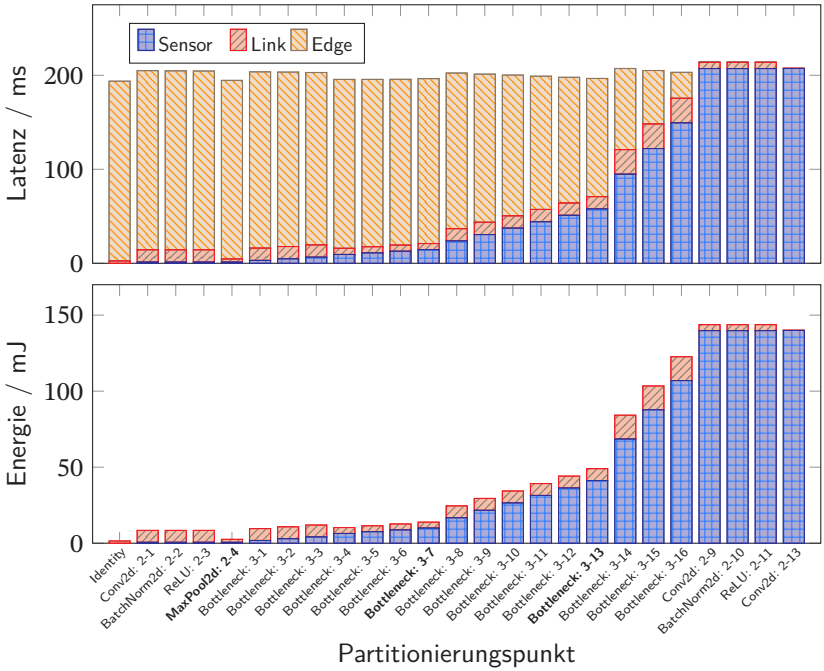


Abbildung 5.3: Evaluation des FCN-ResNet50 für alle potenziellen Partitionierungspunkte mit der Simba-Architektur in der Sensor-Einheit

Falle erstgenannter Metrik ist zu erkennen, dass sich für FCN-ResNet50 nur sehr geringe Unterschiede in der Ausführungszeit zwischen den verschiedenen Aufteilungen ergeben. Die Latenz wird in dieser Systemkonfiguration im Wesentlichen durch den Link beeinflusst. Folglich hängt die Wahl des Partitionierungspunkts insbesondere von dem gewünschten Energiebedarf in der sensornahen Plattform ab.

Hierbei zeigen die Ergebnisse der Vorstudie ein klares Bild. Zu Beginn des CNNs ist der Energiebedarf des Hardwarebeschleunigers in der Sensor-Einheit sehr klein, wodurch die Leistungsaufnahme vom Link dominiert wird. Diese ist wiederum abhängig von der Größe der Output Feature Map, welche in der ersten Hälfte des Netzes ihre lokalen Minima bei den Partitionierungspunkten *Maxpool2d: 2-4* und *Bottleneck: 3-7* aufweist. Soll Arbeitslast von der Edge-Plattform genommen und gleichzeitig der Energiebedarf sowie die benötigte Bandbreite niedrig gehalten werden, empfiehlt sich folglich die Wahl einer der beiden Punkte. Ist dagegen ein höherer Energiebedarf

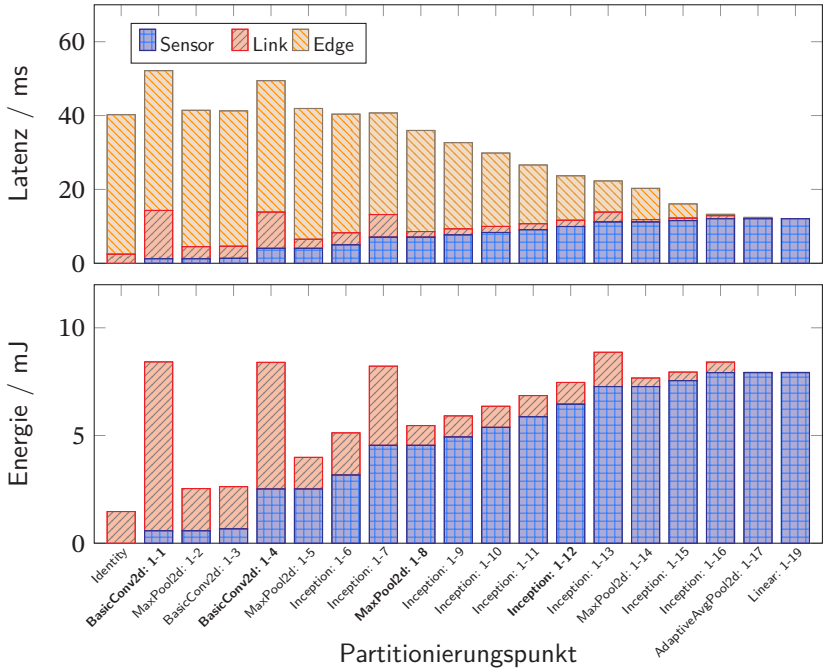


Abbildung 5.4: Evaluation des GoogLeNet für alle potenziellen Partitionierungspunkte mit der Simba-Architektur in der Sensor-Einheit

in der sensornahen Plattform möglich, kann zusätzlich *Bottleneck: 3-13* als möglicher Partitionierungspunkt betrachtet werden. Dieser markiert den letzten Layer im FCN-ResNet50 bevor der Energiebedarf in der Sensor-Einheit stark ansteigt und bietet dadurch eine gute Abwägung zwischen den betrachteten Metriken sowie eine signifikante Übernahme der notwendigen Rechenoperationen.

GoogLeNet

Im Gegensatz zu den Ergebnissen der Partitionierung für FCN-ResNet50, ergibt sich für GoogLeNet, dass der Link deutlich mehr Einfluss auf die Latenz sowie den Energiebedarf des Systems hat (s. Abbildung 5.4). Insbesondere die Partitionierungspunkte *BasicConv2d: 1-1* und *BasicConv2d: 1-4* belegen, dass eine Partitionierung aufgrund der enormen Datenmenge, die mittels Gigabit

Ethernet zwischen den beiden Plattformen übertragen werden muss, zur signifikanten Erhöhung von Latenz und Energiebedarf beitragen kann. Die Nichtberücksichtigung des Links zwischen den beiden Plattformen kann also unter Umständen zu einer klaren Fehleinschätzung der Systemperformance für bestimmte Partitionierungen führen.

Daneben ergibt sich für GoogLeNet eine interessante Konstellation der beiden betrachteten Metriken, da die Latenz mit zunehmender Tiefe ab Layer *MaxPool2d: 1-8* geringer wird, während der Energiebedarf in der sensornahen Plattform stetig zunimmt. In diesem Fall muss also eine Abwägung zwischen den Randbedingungen des Systems und dem Optimierungsziel getroffen werden. Soll die Übertragung der Zwischenergebnisse dabei eine möglichst geringe Latenz aufweisen, ergeben sich die Layer *MaxPool2d: 1-8* bis *Inception: 1-12* als potenziell sinnvolle Kandidaten für eine Partitionierung der CNN-Inferenz.

SqueezeNet V1.1

Als dritte Anwendung wurde abschließend SqueezeNet V1.1 untersucht, welches hinsichtlich der Anzahl an Layern das kleinste CNN im Vergleich zu FCN-ResNet und GoogLeNet ist. Die Ergebnisse der Vorstudie für dieses Netz sind in Abbildung 5.5 dargestellt. Ähnlich wie für GoogLeNet zeigen sich dabei mehrere Schichten wie das *Conv2d: 2-1-* oder *Fire: 2-5*-Layer als offensichtlich nachteilig aufgrund der benötigten Dauer und Energie für die Übertragung der Daten zwischen den beiden Plattformen. Im Vergleich zu den beiden anderen Netzen lohnt sich für SqueezeNet V1.1 dafür die Partitionierung auch hinsichtlich der benötigten Bandbreite. So wird durch die Wahl entsprechender Partitionierungspunkte die Datenmenge im Vergleich zur Übertragung der Rohdaten deutlich verringert und für *MaxPool2d: 2-6* oder *MaxPool2d: 2-9* darüber hinaus ein ähnlich großer Gesamtenergiebedarf erzielt. Soll die Latenz der gesamten Inferenz optimiert werden, stellt zusätzlich *Fire: 2-11* einen potenziellen Partitionierungspunkt dar, da hier das globale Minimum für SqueezeNet V1.1 zu finden ist.

Simulationszeit

Die Ergebnisse der Inferenz-Partitionierung für die drei gewählten Netze zeigen, dass die Aufteilung der Netze auf ASICs sinnvoller Bestandteil im

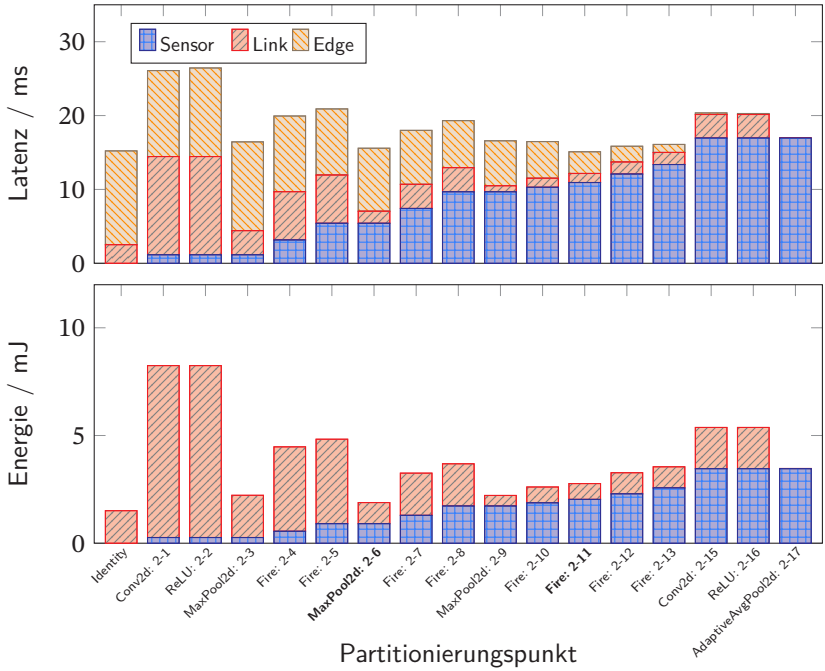


Abbildung 5.5: Evaluation des SqueezeNet V1.1 für alle potenziellen Partitionierungspunkte mit der Simba-Architektur in der Sensor-Einheit

Design-Prozess verteilter Systeme sein sollte. Dafür werden allerdings Simulationsumgebungen benötigt, die in annehmbarer Zeit zuverlässige Abschätzungen der untersuchten Metriken für das Hardware/Software Co-Design liefern. Dies ermöglicht die Untersuchung eines größeren Entwurfsraums, um eine vorteilhafte Systemarchitektur zu finden.

Im Falle der ersten Vorstudie ist dabei insbesondere die Simulationszeit für den sensornahen Knoten relevant, da Timeloop hierfür zunächst ein passendes Mapping für den Beschleuniger finden muss, bevor eine Evaluation der Latenz und des Energiebedarfs erfolgen kann. Allerdings muss auch die Zeit für die Evaluation der Edge-Plattform mittels Messung berücksichtigt werden. Um sicherzustellen, dass die ermittelten Werte für die Evaluationsdauer der Inferenz-Partitionierung zuverlässig sind, wurden für diese Untersuchung insgesamt 50 Simulationen unter gleichen Bedingungen durchgeführt und der Median ausgewählt. Für diese Betrachtung ist der Link irrelevant, da es sich hierbei um ein rein mathematisches Modell handelt, welches die entspre-

Architektur	Partitionierungs-Punkte	Sensor-Einheit (Sim.-Zeit)	Edge-Einheit (Messdauer)
FCN-ResNet50	25	610,336 s	3319,526 s
GoogLeNet	19	409,579 s	395,256 s
SqueezeNet V1.1	17	167,015 s	53,998 s

Tabelle 5.1: Mediane Simulationszeit zur Evaluation verschiedener CNN-Architekturen aus jeweils 50 Läufen

chenden Metriken in weniger als einer Millisekunde für die Evaluation des Gesamtsystems zurückgibt. Die ermittelten Werte, dargestellt in Tabelle 5.1, basieren auf der Untersuchung mit einer Simba-Architektur in der sensornahen Plattform und einer NVIDIA Jetson TX-2 GPU als Beschleuniger in der Edge-Einheit.

Wie zu erwarten, steigt die Simulationszeit mit der Größe des CNN an, wobei die Evaluation der sensornahen Plattform für GoogLeNet und SqueezeNet V1.1 noch mehr Zeit beansprucht als die Messungen der Latenz in der Edge-Plattform. Die Ergebnisse zeigen darüber hinaus, dass eine Evaluation basierend auf realen Messungen für größere Netze wie dem FCN-ResNet50 deutlich mehr Zeit beansprucht als im Falle der Simulation mittels Timeloop. Grund dafür ist die hohe Anzahl an notwendigen Messungen, welche notwendig ist, um einen möglichst realitätsnahen Median-Wert für die Ausführungsdauer der Inferenz zu ermitteln.

Abgesehen davon liegen die Simulationszeiten mittels Timeloop mit maximal 610,34 s in einem zeitlichen Rahmen, der für die Exploration verschiedener Partitionierungspunkte eines CNN in der Entwurfsphase unproblematisch ist. Folglich weist diese erste Vorstudie nach, dass die Verwendung von Simulationen für die Untersuchung der Inferenz-Partitionierung gut skaliert und somit neben der hohen Flexibilität im Vergleich zu Messungen auch in dieser Hinsicht einen klaren Vorteil bietet.

5.2 Vorstudie II: Partitionierung in FPGA-basierten Systemen

Neben dem Einsatz von ASICs als Hardwareplattform in eingebetteten Systemen, können auch FPGAs für die lokale Vorverarbeitung der Sensordaten genutzt werden. Diese sind zwar nicht so performant und energieeffizient wie als ASIC implementierte Hardwarebeschleuniger, bieten dagegen aber eine hohe Flexibilität aufgrund der Rekonfigurierbarkeit. Aufgrund der rasanten Entwicklungen der Topologien von DNNs, erlaubt diese Eigenschaft folglich eine höhere erwartbare Lebenszeit, da Anpassungen an der Architektur des Hardwarebeschleunigers zur Laufzeit möglich sind. Darüber hinaus ist die Entwicklungszeit für Beschleuniger auf einem FPGA signifikant kürzer als für ein ASIC, wodurch sich auch in diesem Bezug der Einsatz von eingebetteten FPGAs lohnt. Wie in Abbildung 5.6 dargestellt, wurde in der zweiten Vorstudie die Inferenz-Partitionierung unter der Verwendung eines eingebetteten FPGAs in Kombination mit einer GPU untersucht. Im Gegensatz zur in Abschnitt 5.1 gewählten Systemarchitektur, erfolgt die Ausführung der Layer des DNN auf dem FPGA in einer Pipeline, sodass eine parallele Ausführung stattfindet. Folglich ergibt sich ein deutlich höherer Datendurchsatz, wovon insbesondere sicherheitskritische Anwendungen, beispielsweise in Fahrassistenzsystemen, profitieren. Die Ergebnisse dieser Vorstudie sowie die Erläuterungen wurden durch den Autor in [Kre23a] veröffentlicht.

5.2.1 Toolchain

Die Analyse der Inferenz-Partitionierung für die zuvor beschriebene Systemkonfiguration erfordert erneut keine exakten Metriken, sondern kann auf Abschätzung der relevanten Metriken basieren. Dabei liegt der Fokus in dieser zweiten Vorstudie nicht auf Latenz oder Energiebedarf, sondern auf der Verringerung der benötigten Bandbreite zwischen FPGA und GPU sowie der Auslastung der Logikzellen im FPGA. Letztere stellen bei der Realisierung der Inferenz-Pipeline in Hardware eine wesentliche Einschränkung dar. Grund dafür ist, dass jeder Layer eines DNN für sich viele Ressourcen in Anspruch nimmt, wodurch die vollständige Implementierung größerer Modelle insbesondere auf eingebetteten FPGAs unmöglich ist. Folglich müssen Teile der Inferenz auf eine zweite Plattform ausgelagert werden, im Falle dieser Vorstudie eine GPU. Da die Bandbreite der Schnittstelle zwischen diesen Systemen in der Regel begrenzt ist, muss diese Metrik in die Analyse der Inferenz-Partitionierung einbezogen werden.

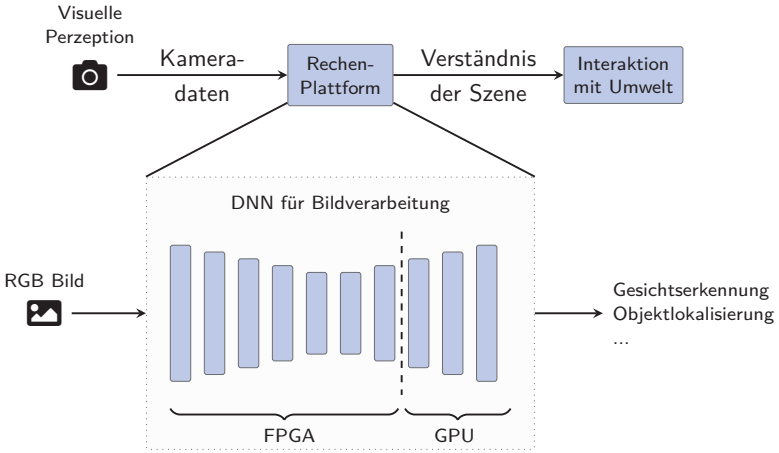


Abbildung 5.6: Aufteilung der DNN Inferenz zwischen FPGA und GPU

Für die Evaluation wurde eine Toolchain entwickelt, welche die begrenzten Hardwareressourcen sowie die benötigte Bandbreite zur Übertragung der Zwischenergebnisse in die Suche nach einem vorteilhaften Partitionierungspunkt einbezieht. Diese ist in Abbildung 5.7 dargestellt. Basierend auf der Annahme, dass die Inferenz-Pipeline im FPGA performanter ist als die GPU, sucht die Toolchain so nach einem optimalen Partitionierungspunkt mit möglichst geringen Anforderungen an die Link-Bandbreite.

Dafür benötigt diese neben dem DNN Modell die Spezifikationen des eingebetteten FPGA bezüglich der verfügbaren Hardwareressourcen sowie vom Nutzer festgelegte Systembeschränkungen hinsichtlich Hardwareauslastung und der maximal verfügbaren Bandbreite. Diese ist besonders in Systemen bestehend aus mehreren Sensoren, welche über denselben Bus mit der zentralen Rechenplattform verbunden sind, eine relevante Einschränkung, um auch zusätzlichem Datenverkehr gerecht zu werden.

In einem ersten Schritt führt die Toolchain eine Quantisierung der Gewichte sowie der Output Feature Map innerhalb eines QAT durch, um die Komplexität der Berechnungen sowie den Speicherverbrauch zu reduzieren. Das quantisierte DNN wird anschließend schichtweise hinsichtlich der benötigten Hardwareressourcen analysiert. Dabei fließen in die Untersuchung auch Informationen über das Vorhandensein spezieller Intellectual Property (IP)-Blöcke im FPGA ein, wie beispielsweise BRAMs oder DSPs. Für die Suche nach einem optimierten Partitionierungspunkt muss zusätzlich die jeweils benö-

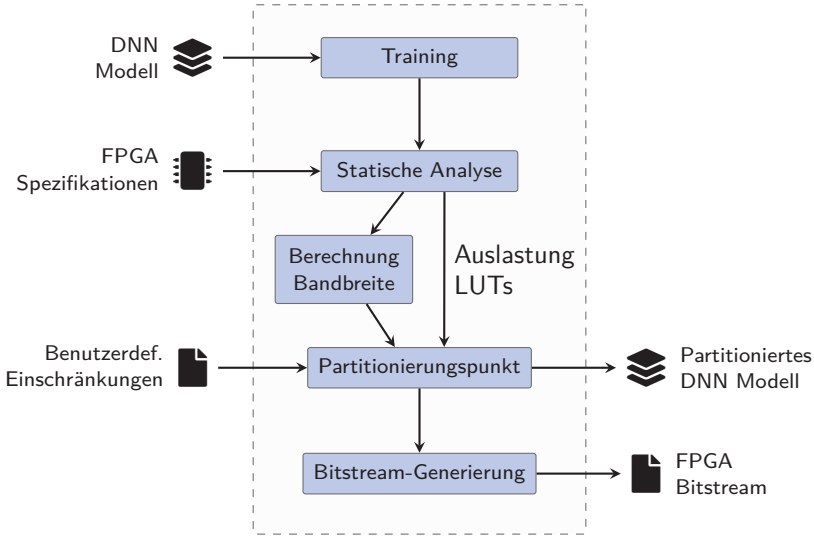


Abbildung 5.7: Übersicht der Toolchain für die zweite Vorstudie

tigte Link-Bandbreiten anhand der Output Feature Maps der Layer berechnet werden. Mittels dieser Informationen gibt die Toolchain schließlich einen Partitionierungspunkt aus und generiert außerdem einen entsprechenden Bitstream für den FPGA.

5.2.2 Training und statische Analyse

Ähnlich der Vorgehensweise in Vorstudie I, ist auch für die Inferenz im eingebetteten FPGA eine Optimierung des DNNs notwendig, um eine energieeffiziente und performante Hardwareimplementierung zu ermöglichen. In diesem Fall wurde dabei auf das Open-Source Tool Brevitas [153] zurückgegriffen, welches diesen Schritt automatisiert durchführt. Um eine signifikante Reduktion der Hardwareressourcen bei geringem Verlust der Genauigkeit des DNNs zu erzielen, führt das Tool eine layerweise Quantisierung mit anschließendem QAT durch. Aus den Ergebnissen wird schließlich ein optimiertes DNN für die Inferenz-Partitionierung abgeleitet.

Da in dieser Vorstudie ein AMD FPGA als Zielplattform zum Einsatz kam, bildet das von AMD entwickelte FINN Framework [154] einen zentralen Bestandteil der Toolchain. Im ersten Schritt führt dieses Framework eine Ent-

wurfsraumexploration zur Abschätzung der benötigten Hardwareressourcen sowie der Performance der DNN Inferenz durch. Anhand der Ergebnisse wird in der nächsten Stufe eine Codegenerierung für die High-Level Synthese (HLS) durchgeführt, welche die Hardwareblöcke für die Inferenz entsprechend der quantisierten DNN Architektur abbildet. Anschließend lässt sich mit dem FINN Framework zusätzlich ein Bitstream für den verwendeten FPGA mittels AMD Vivado generieren.

5.2.3 Methodik

Obwohl FINN bei der Suche nach einer optimierten Hardwareimplementierung verschiedene Metriken betrachtet, findet die Größe des FPGAs hinsichtlich verfügbarer Hardwareressourcen innerhalb des Frameworks keine Berücksichtigung. Folglich kann die Synthese der Logik auf den FPGA scheitern, wenn die DNN Inferenz-Pipeline mehr LUTs oder FFs als vorhanden benötigt. An dieser Stelle ist also ein zusätzliches Modul innerhalb der Toolchain erforderlich, welches die Auslastung des FPGAs bei der Suche nach einem potenziellen Partitionierungspunkt berücksichtigt.

Als Grundlage für die Ermittlung der benötigten Hardwareressourcen pro Layer dienen die Schätzungen des FINN Frameworks, die während der HLS auf verschiedenen Abstraktionsebenen automatisch erstellt werden. Eine erste Schätzung erfolgt dabei bereits vor der IP-Block-Generierung, welche allerdings sehr ungenaue Angaben liefert. Dagegen erzielt die Out-of-Context Synthese präzise Ergebnisse, da in diesem Schritt bereits Hardwareoptimierungen durchgeführt werden. Dies führt allerdings zu einer signifikanten Erhöhung der Simulationszeit. Da diese Vorstudie nicht auf eine hohe Präzision der Ergebnisse abzielt, sondern die generelle Sinnhaftigkeit der Inferenz-Partitionierung auf FPGAs im Fokus steht, wird daher auf die Abschätzungen nach erfolgreicher IP-Block-Generierung zurückgegriffen. Dieser Ansatz stellt folglich einen guten Kompromiss zwischen Simulationszeit und Genauigkeit der Ergebnisse dar. Neben der Betrachtung der benötigten Hardwareressourcen im FPGA ist für die Inferenz-Partitionierung, wie bereits in Vorstudie I nachgewiesen, die benötigte Link-Bandbreite zwischen den Plattformen eine relevante Metrik. Diese lässt sich im Gegensatz zur zeitaufwendigen Generierung der IP-Blöcke mittels des FINN-Frameworks in sehr kurzer Zeit mathematisch anhand der Output Feature Map des Layers am Partitionierungspunkts exakt bestimmen.

Basierend auf den beiden ermittelten Metriken wird im nächsten Schritt die Suche nach einem optimierten Partitionierungspunkt der DNN Inferenz

Pseudocode 1 Algorithmus zur Bestimmung eines optimierten Partitionierungspunkts auf einem FPGA

```
1:  max_layer ← letztes DNN-Layer welches auf die Hardware passt
2:  part_pnt ← max_layer
3:  for each layer in (max_layer, first_layer) do
4:      bw_ratio ← bandwidth[part_pnt] / bandwidth[layer]
5:      hw_ratio ← hw_resources[part_pnt] / hw_resources[layer]
6:      if bw_ratio > 1 and bw_ratio/hw_ratio > threshold_ratio then
7:          part_pnt ← layer
8:      end if
9:      if stop condition fulfilled then
10:         break
11:     end if
12: end for
13: return part_pnt
```

zwischen FPGA und GPU durchgeführt. Dabei müssen zusätzlich die begrenzten Hardware-Ressourcen in Bezug auf die Anzahl der LUTs und FFs und, falls vorhanden, der DSP- und BRAM-Blöcke berücksichtigt werden, um das partitionierte DNN auf dem FPGA zu synthetisieren. Der im Rahmen dieser Vorstudie entwickelte Suchalgorithmus wird in Pseudocode 1 beschrieben. Darin wird zunächst der Partitionierungspunkt ermittelt, welcher zu einer maximalen Auslastung der Hardwareressourcen auf dem FPGA führt. Anschließend erfolgt ausgehend von ebenjenem Layer eine rückwärts gerichtete iterative Suche nach einem Partitionierungspunkt mit minimaler Bandbreite und gleichzeitig hohem Ressourcenverbrauch. Hierfür müssen zuvor zwei Parameter definiert werden.

Das Abbruchkriterium *stop condition* legt die minimal zulässige Hardwareauslastung des FPGAs fest. Sobald die Suche diesen Wert unterschreitet, wird sie abgebrochen. Zusätzlich definiert der Schwellwert *threshold_ratio* prozentual das minimale Verhältnis zwischen der Verbesserung der Bandbreitenanforderungen und der Verringerung der Hardwareressourcen im Vergleich zum aktuell besten Partitionierungspunkt. Übersteigt der Layer diesen Wert, gilt er als neuer optimierter Punkt, bis eine bessere Lösung gefunden bzw. das Abbruchkriterium erfüllt ist. Damit bildet der Algorithmus die Annahme ab, dass die Inferenz auf dem FPGA schneller ausgeführt wird als auf der GPU. Folglich ergibt sich das Optimierungsziel, möglichst viele Teile des DNN auf dem FPGA bei geringer Link-Bandbreite zwischen den Hardwareplattformen zu berechnen. Im Anschluss an die Suche nach einem optimierten Partitionierungspunkt teilt die Toolchain das DNN entsprechend in zwei Teile auf und generiert daraus einen Bitstream für den FPGA mittels AMD Vivado HLS bzw. eine ONNX Datei zur Ausführung des Subnetzes auf der GPU.

Plattform	LUTs	FFs	BRAMs	DSPs
ZedBoard	53,200	106,400	280	220
Ultra96-V2	70,560	141,120	432	360
ZCU104	230,400	460,800	624	1,728

Tabelle 5.2: Verfügbare Hardwareressourcen der evaluierten MPSoCs

5.2.4 Evaluation

Für die Evaluation der entwickelten Toolchain kamen drei verschiedene FPGA-basierte Entwicklungsboards zum Einsatz (ZedBoard, Avnet Ultra96-V2 und AMD Zynq UltraScale+ MPSoC ZCU104), welche sich in der Anzahl an verfügbaren Hardwareressourcen unterscheiden. Dieser Ansatz erlaubt die Untersuchung der Inferenz-Partitionierung für unterschiedliche Anwendungsbereiche, von IoT Plattformen zu leistungsfähigen Zentralrechnern. Die vorhandenen Logikblöcke der verwendeten MPSoCs sind in Tabelle 5.2 aufgelistet. In dieser Vorstudie wurde zur Evaluation ein Intel Quad-Core i7-8565U mit Ubuntu 18.04 genutzt, wobei ein von AMD bereitgestellter Docker-Container für das FINN-Framework die Toolchain ausführte. Die Abschätzung der benötigten Hardwareressourcen erfolgte mittels AMD Vivado, Version 2020.1.

Test-Anwendungen

Zur Untersuchung der Einflüsse der Inferenz-Partitionierung auf die Performance wurde ein MobileNet V1 verwendet [155]. Dieses zeichnet sich durch gute Genauigkeit (70.6 % für ImageNet) bei geringer Komplexität der Architektur (569 Millionen MAC Operationen und 4,2 Millionen trainierbare Parameter) aus, was auf die dreizehn Depthwise-CONV Blöcke zurückzuführen ist. Aufgrund der genannten Eigenschaften eignet sich das MobileNet V1 sehr gut für den Einsatz in eingebetteten System. Für die nachfolgende Analyse der Partitionierung wurde ein vortrainiertes Modell mit auf vier Bits quantisierten Gewichten und Aktivierungen verwendet.

Ergebnisse

Die Ausführung der Pipeline-Inferenz bedarf der Abbildung der einzelnen Layer eines DNNs auf grundlegende Hardwarebausteine, welche anschlie-

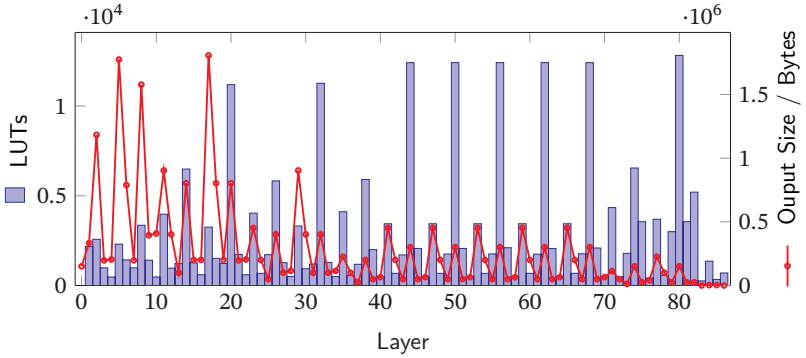


Abbildung 5.8: Ergebnisse der zweiten Vorstudie

ßend auf den FPGA synthetisiert werden. Im Falle des MobileNet V1 ergeben sich bei der Nutzung des FINN Frameworks insgesamt 86 zu synthetisierende Layer. Auch in diesem Fall spielt für die Anwendbarkeit der Toolchain in der Forschung und Entwicklung die Gesamtlaufzeit eine wichtige Rolle. Für das Einlesen des zuvor mittels Brevitas generierten ONNX-Graphen und die Durchführung der HLS benötigt das FINN Framework für das MobileNet V1 etwa 81,6 min. Wie zu erwarten, entfällt mit ca. 79 min (97 % der Laufzeit) der größte Anteil auf die Generierung der IP-Blöcke. Da dieser Schritt allerdings pro quantisiertem DNN Modell nur bei der allerersten Evaluation notwendig ist und dessen Ergebnisse bei jeder weiteren Untersuchung wiederverwendet werden können, liegt die gemessene Laufzeit der Toolchain auf dem verwendeten SoC in einem annehmbaren Bereich.

Abbildung 5.8 zeigt die Ergebnisse der IP-Block Generierung sowie die Output Size der einzelnen Layer für MobileNet V1. Dabei lässt sich erkennen, dass die Größe der Aktivierungen tendenziell zum letzten Layer hin geringer wird. Dagegen zeigen die Ergebnisse aber auch, dass der Hardwareverbrauch auf dem FPGA gegen Ende des DNN zunimmt, was zu einem deutlichen Sprung in den Anforderungen an die vorhandenen Ressourcen führt. Aus diesem Grund ist bei der Suche nach einem optimierten Partitionierungspunkt eine Abwägung zwischen diesen beiden Metriken erforderlich. In diesem Kontext sind die durch den Nutzer der Toolchain vorgegebenen Einschränkungen zu berücksichtigen, welche sich aus der Hardwareplattform sowie dem verwendeten Link ergeben.

Im Fokus der Exploration steht die Hardwareressourcennutzung, welcher in eingebetteten Systemen aufgrund der begrenzten Chipfläche eine hohe

Plattform	nur FFs/LUTs				alle Ressourcen			
	max. Layer	Part. Punkt	BW Red. (%)	HW Red. (%)	max. Layer	Part. Punkt	BW Red. (%)	HW Red. (%)
ZedBoard	22	21	6,78	1,19	19	19	0	0
Ultra96-V2	31	25	50	18,73	25	25	0	0
ZCU104	79	73	50	7,88	79	73	50	6,38

Tabelle 5.3: Ergebnisse der zweiten Vorstudie

Bedeutung zukommt. Um verschiedene Szenarien zu simulieren, untersucht diese Vorstudie die in Tabelle 5.2 genannten FPGA-Plattformen. Zusätzlich wird evaluiert, wie sich die Inferenz-Partitionierung auf ein System ohne IP-Blöcke, also ohne DSPs sowie BRAMs, und folglich lediglich bestehend aus FFs und LUTs auswirkt. Für die Experimente wurde das Abbruchkriterium auf mindestens 70 % Ressourcenverbrauch festgelegt, um eine möglichst hohe Latenz zu erzielen. Eine vollständige Auslastung der Hardware ist in einem FPGA aufgrund der limitierten internen Routing-Möglichkeiten zwischen den Blöcken nicht möglich. Aus diesem Grund handelt es sich bei den in Tabelle 5.3 dargestellten Ergebnissen um hypothetische Werte, die lediglich eine Tendenz anhand der unterschiedlich großen FPGAs zeigen.

Dabei ergibt sich für das ZedBoard eine maximale Anzhal von 22 Layern, die auf die Hardwareressourcen abgebildet werden könnten. Die Toolchain gibt hierbei das 21. Layer als bevorzugten Partitionierungspunkt aus, wenn nur FFs und LUTs zum Einsatz kommen sollen. Insgesamt führt dies zu einer Reduktion der benötigten Hardwareressourcen von 1,19 %, während die benötigte Bandbreite um 6,78 % sinkt. Bei einem Vergleich des vorliegenden Ergebnisses mit den in Abbildung 5.8 dargestellten Hardwareanforderungen und Bandbreiten pro Layer wird ersichtlich, dass eine Partitionierung nach Layer 20 zwar zu einer Reduktion der benötigten Hardwareressourcen führt, jedoch mit einer signifikanten Zunahme der zu übertragenden Datenmenge zwischen FPGA und GPU einhergeht.

Eine deutliche höhere Reduktion der Bandbreite durch die Toolchain ergibt sich für das Ultra96-V2 und das ZCU104. Hierbei wird eine um 50 % geringere Auslastung des Links im Vergleich zur Implementierung der maximal möglichen Anzahl an Layern im FPGA erzielt. Im Falle des Ultra96-V2 führt die Wahl dieses Partitionierungspunkts dabei zu einer deutlichen Reduktion der benötigten Hardwareressourcen um 18,73 %, wohingegen dies für das ZCU104 mit 7,78 % nicht ganz so signifikant ausfällt.

Bei der Betrachtung der Ergebnisse für den Fall, dass auch die vorhandenen

IP-Blöcke im FPGA von der Toolchain berücksichtigt werden, offenbart sich ein etwas anderes Bild. Lediglich FPGAs von der Größe eines ZCU104 profitieren in diesem Fall hinsichtlich der Hardwareressourcen und der Reduktion der Bandbreite. Grund dafür ist, dass insbesondere kleine FPGAs stark in der Zahl der vorhandenen BRAM-Blöcke und DSPs beschränkt sind. Da folglich eine geringere Menge an Layern auf einem ZedBoard oder Ultra96-V2 implementiert werden können, ergibt sich in beiden Fällen kein Potenzial zur Reduzierung der benötigten Bandbreite. Dagegen ergab die Evaluation für das MobileNet V1 auf einem ZCU104 denselben Partitionierungspunkt wie bei der Betrachtung von ausschließlich FFs und LUTs.

Zusammenfassend ergibt sich anhand dieser zweiten Vorstudie, dass auch FPGA-basierte Systeme von einer Inferenz-Partitionierung profitieren. So lässt sich bereits durch das Auslagern weniger Layer eine signifikante Reduzierung der benötigten Link-Bandbreite erzielen. Aufgrund der eingeschränkten Möglichkeiten des FINN-Frameworks hinsichtlich der Exploration zur Entwurfszeit, fokussiert sich die vorliegende Arbeit im Folgenden allerdings auf ASIC-basierte Systeme. Dennoch zeichnet sich das im weiteren Verlauf vorgestellte Framework CNNParted durch eine hohe Flexibilität aus, sodass durch die Integration einer geeigneten Schnittstelle zu FINN auch die Analyse von FPGA-basierten Systemen ermöglicht werden kann.

5.3 Allgemeine Problemdefinition

Nachdem mittels der beiden Vorstudien der Nachweis erbracht werden konnte, dass die Inferenz-Partitionierung vorteilhaft für ein eingebettetes System hinsichtlich funktionaler und nicht-funktionaler Metriken ist, erfolgt im nächsten Schritt die Entwicklung einer generischen Methodik für die Suche nach einem optimierten Partitionierungsschema. Dafür ist zunächst eine eindeutige Definition des Problems notwendig.

Als Basis dient die Annahme, dass jedes DNN als gerichteter Graph $G = (L, E)$ mit den Layern L als Knoten und den entsprechenden Verbindungen zwischen diesen als Kanten E dargestellt werden kann. Heutzutage finden sich in vielen DNNs parallele Verzweigungen oder auch Skip-Connections wieder, um das sogenannte *Vanishing Gradient* Problem zu minimieren. Dies tritt insbesondere bei großen Netzwerken auf, wodurch während der Backpropagation die Gradienten ab einem gewissen Layer nur noch verschwindend klein sind und somit zu keiner Änderung der Gewichte in den ersten Schichten

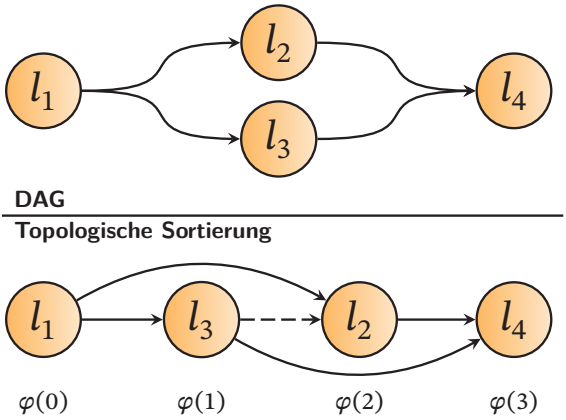


Abbildung 5.9: Topologische Sortierung eines gerichteter azyklischer Graph (DAG)

des DNNs führen. Um folglich auch solche DNN-Architekturen mit parallelen Verzweigungen bestmöglich zu partitionieren, muss dies während der Exploration Berücksichtigung finden. Genauer bedeutet dies für die Inferenz-Partitionierung, dass der Knoten an einem Schnittpunkt im Graph mehr als nur eine Kante als Eingabe haben kann. Infolgedessen vergrößert sich der Entwurfsraum im Vergleich zu rein sequenziellen DNNs erheblich, da es für solche Modelle mehr als nur eine mögliche sequenzielle Verarbeitungsreihenfolge der Layer gibt und dabei sicherzustellen ist, dass alle Vorgängerknoten einer Schicht bereits ausgeführt wurden.

Nicht-rekurrente DNNs sind azyklisch und lassen sich folglich als DAG darstellen. Damit lässt sich eine topologische Sortierung vornehmen, welche eine sequenzielle Liste an nacheinander auszuführenden Layern ausgibt und dabei garantiert, dass Datenabhängigkeiten erfüllt sind. Genauer handelt es sich um eine lineare Anordnung der Knoten eines DAGs [156], wie beispielhaft in Abbildung 5.9 skizziert. Dabei ist ein kleines DNN bestehend aus vier Layern $(l_1, \ldots, l_4)$ dargestellt, wobei die beiden Schichten l_2 und l_3 den gleichen Vorgänger und Nachfolger haben. Demnach kann die Ausführung dieser beiden Layer entweder parallel oder sequenziell erfolgen. Wie in Abbildung 5.10 dargestellt, generiert die topologische Sortierung eine mögliche sequenzielle Verarbeitungsreihenfolge, wobei die Wahl, welcher der beiden Knoten zuerst gewählt wird, von der Implementierung des Algorithmus abhängt. In diesem Beispiel wird somit l_3 vor l_2 ausgeführt. Basierend darauf lässt sich die topologische Sortierung wie folgt definieren:

Definition 2 (Topologische Sortierung)**.** *Eine topologische Sortierung eines DNNs bestehend aus einer Menge L an m Layern ist eine konsistente Abfolge dieser Schichten und ist gegeben durch* $\varphi : L \to \{1,\dots,m\}$*, wobei*

$$\forall l_1, l_2 \in L : \ \varphi(l_1) < \varphi(l_2) \Rightarrow l_1 \textit{ wird vor } l_2 \textit{ ausgeführt},$$

$$\forall l_1, l_2 \in L, \ l_1 \neq l_2 \Rightarrow \varphi(l_1) \neq \varphi(l_2).$$

Unter Verwendung dieser topologischen Sortierung erfolgt so im nächsten Schritt die Suche nach einem validen Partitionierungsschema. Aufbauend auf der Beschreibung in Abschnitt 5.1 kann ein Partitionierungspunkt allgemein wie folgt definiert werden:

Definition 3 (Partitionierungspunkt)**.** *Ein Partitionierungspunkt ist ein Layer* $l_s \in L$ *mit* $s \in \{1,\dots,|L|\}$ *in einem partitionierten DNN bestehend aus der Menge P von sequenziell ausgeführten Partitionen, sodass*

$$\psi : \varphi \to \{1,\dots,|P|\}, \ \forall l_1, l_2 \in L : \ \varphi(l_1) < \varphi(l_2) \Rightarrow \psi(\varphi(l_1)) \leq \psi(\varphi(l_2)),$$

$$\psi(\varphi(l_s)) \neq \psi(\varphi(l_{s+1}))$$

Hierbei beschreibt ψ die sequenzielle Zuordnung der einzelnen Schichten einer gegebenen topologischen Sortierung φ auf die Partitionen. Infolgedessen entspricht ein Partitionierungspunkt l_s dem letzten ausgeführten DNN-Layer einer Partition.

Für die Suche nach einem validen Partitionierungsschema müssen somit Partitionierungspunkte im gegebenen DNN bestehend aus $|L|$ Layern gefunden und die resultierenden Partitionen auf eine Beschleunigerplattform $a \in A$ gemappt werden. In diesem Zusammenhang ist insbesondere die maximal zulässige Anzahl an Partitionen relevant für die Suche. Dies ist definiert als $p \leq |L|$. Daraus lässt sich für die Suche nach einem validen Partitionierungsschema folgende Definition des Algorithmus zur Exploration des Entwurfsraums ableiten:

Definition 4 (Explorationsalgorithmus)**.** *Der Explorationsalgorithmus ist ein automatisiertes Verfahren, welches als Eingabe*

1. *ein DNN beschrieben durch L und* φ*,*
2. *eine Menge an spezifizierten Beschleunigerplattformen A, und*
3. *die maximale Anzahl an Partitionen* $p \in \mathbb{N}$

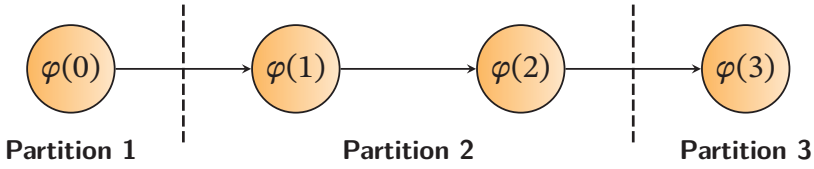


Abbildung 5.10: Exemplarisches Mapping von DNN-Layer auf Partitionen

verwendet und ein optimiertes Hardware-Mapping des DNNs auf die vorhandenen Beschleunigerplattformen generiert. Dieses Mapping ist gegeben durch

1. *eine lineare Anordnung von $p' \leq p$ generierten Partitionen,*
2. *einen Vektor $\vec{\omega} \in \{1, \ldots, |L|\}^{p'}$, welcher das Mapping der gegebenen DNN-Layer auf die generierten Partitionen beschreibt, und*
3. *einen Vektor $\vec{\sigma} \in \{1, \ldots, |A|\}^{p'}$, welcher das Mapping der generierten Partitionen auf die verfügbaren Beschleunigerplattformen beschreibt.*

Wie in Abbildung 5.10 dargestellt, unterteilt das generierte Mapping die topologische Sortierung in p' Partitionen. Dabei entspricht jede Partition einer zusammenhängenden Folge von DNN-Schichten, wobei jedes DNN-Layer genau einer Partition zugeordnet ist. Darüber hinaus sind die Partitionen derart angeordnet, sodass für alle $i = 1, \ldots, p' - 1$ das letzte Layer einer Partition i dem ersten DNN-Layer der Partition $i + 1$ vorausgeht. Folglich handelt es sich bei der letzten letzte Schicht der Partition i gemäß Definition 3 einem Partitionierungspunkt.

Um das Mapping zu beschreiben, spezifiziert $\vec{\omega}$ das erste DNN-Layer jeder Partition. Genauer gilt für jedes $i = 1, \ldots, p'$, dass der Wert von ω_i dem φ-Index des ersten Layers entspricht, welcher auf die Partition i gemappt wurde. Aus diesem Grund trifft $\omega_1 = 1$ auf jedes generierte Hardware-Mapping zu. Analog dazu beschreibt $\vec{\sigma}$ das Mapping der Partitionen auf die vorhandenen Beschleunigerplattformen. So gilt für jedes $i = 1, \ldots, p'$, dass der Wert von σ_i einen Index darstellt, welcher die entsprechende Plattform $a \in A$ zur Ausführung der Layer der Partition definiert. In anderen Worten ausgedrückt bedeutet dies, dass alle auf eine Partition i gemappte DNN-Layer auf der Beschleunigerplattform mit dem Index σ_i ausgeführt werden.

Als Teil der Suche nach einem optimierten Partitionierungsschema ist abschließend die Evaluation valider Aufteilungen des DNNs über die verfügbaren Beschleunigerplattformen hinsichtlich verschiedener Metriken notwendig. Daraus leitet sich das Ziel der Exploration ab. Es wird nach

einer Pareto-Menge $P(X)$ mit $\vec{x} \in X$ als geordnetes Tupel $\vec{x} = (\vec{\omega}; \vec{\sigma})$ gesucht, welche das Partitionierungsschema für eine gegebene topologische Sortierung φ und einer Menge an Layern L hinsichtlich funktionaler und nicht-funktionaler Metriken optimiert. Für eine ressourcenbasierte Betrachtung der Inferenz-Partitionierung ist letzteres unabdingbar. In Bezug auf DNNs ist die Genauigkeit die wesentliche funktionale Metrik, deren Wert insbesondere durch Quantisierung oder approximatives Rechnen beeinflusst wird. Im Kontext eingebetteter Systeme sind die maßgeblichen nicht-funktionalen Metriken Latenz, Energiebedarf, Datendurchsatz, Flächenverbrauch und Linkbandbreite. Diese sollten daher in der Exploration entsprechend berücksichtigt werden, was zu der folgenden Beschreibung der Optimierungsstrategie führt:

Definition 5 (Mehrzieloptimierung)**.** *Ausgehend von der Entscheidungsvariable $\vec{x}$ besteht das Ziel der Mehrzieloptimierung darin, eine Pareto-Front zu finden, wobei Latenz $d(\vec{x})$, Energiebedarf $e(\vec{x})$, Flächenverbrauch $s(\vec{x})$ und Linkbandbreite $u(\vec{x})$ zu minimieren sowie Datendurchsatz $th(\vec{x})$ und Top-1-Genauigkeit $acc(\vec{x})$ zu maximieren sind.*

Im Gegensatz zur Nutzung einer festgelegten Kostenfunktion, welche die genannten Metriken entsprechend der Systemanforderungen gewichtet, ermöglicht die Pareto-Optimierung eine generische Suche nach vorteilhaften Partitionierungsschemata. Die Ergebnisse der Exploration können so einerseits allgemeingültig für ein gegebenes DNN in Kombination mit den definierten Beschleunigerplattformen interpretiert und andererseits entsprechend für einen bestimmten Anwendungsfall mit klar definierten Anforderungen ausgelegt werden.

5.4 CNNParted Framework

Auf Grundlage der allgemeinen Problemdefinition lässt sich eine entsprechende Methodik für die hardwareorientierte Inferenz-Partitionierung ableiten und realisieren. Dabei kann die Umsetzung je nach Anforderungen grundsätzlich auf unterschiedlichen Ebenen erfolgen. Wie bereits in Abschnitt 5.1 erwähnt, liefern zyklenakkurate Simulationen in der Regel recht genaue Abschätzungen relevanter Metriken. Allerdings sind diese insbesondere im Kontext großer DNNs nicht praktikabel, da die Simulationszeit in diesen sehr hoch ausfällt. Daher wurde hier ein Ansatz auf Systemebene gewählt, welcher zwar quantitativ schlechtere Abschätzungen liefert, dafür aber bedeutend schneller abläuft. Somit erfolgt bereits zur Entwurfszeit ein umfangreiches

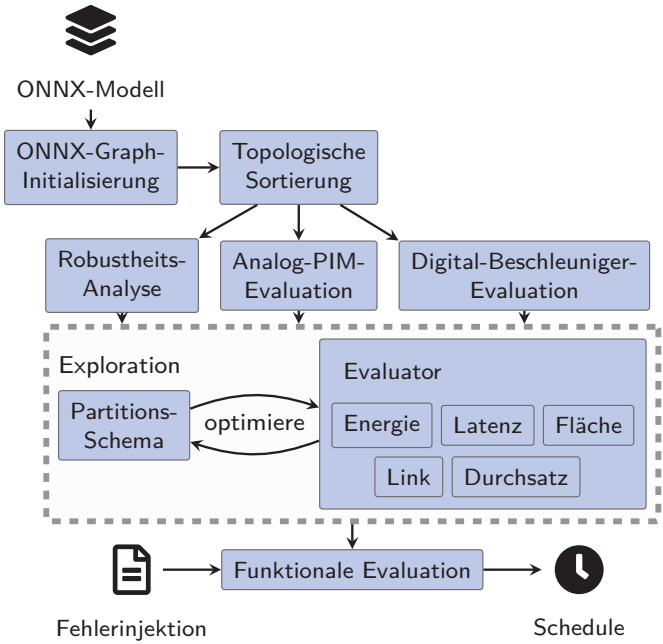


Abbildung 5.11: Übersicht des CNNParted Frameworks

Hardware/Software Co-Design, da verschiedene Systemkonfigurationen und Aufgaben hinsichtlich des Zusammenspiels untersucht und anschließend bewertet werden. Für die Bewertung sind dabei keine präzisen Abschätzungen, sondern lediglich Tendenzen relevant, um Systemarchitekturen zu erkennen, die sich für den gegebenen Anwendungsfall besonders gut bzw. schlecht eignen. Das entwickelte Framework namens CNNParted stellt die Implementierung eines solchen Ansatzes auf Systemebene mit dem Fokus auf ASICs dar und wird nachfolgend detailliert beschrieben. Aufgrund der geringeren Flexibilität von FPGA-Frameworks für DNNs wie dem in Abschnitt 5.2 verwendeten AMD FINN, finden diese nachfolgend keine Berücksichtigung. Dennoch wäre eine Erweiterung des vorgestellten Verfahrens um FPGA-basierte Beschleunigerplattformen in Zukunft denkbar. Die nachfolgenden Erläuterungen zu CNNParted sind in einigen Teilen den Publikationen des Autors [Kre23b, Kre24a, Kre25] entnommen.

Abbildung 5.11 zeigt den Aufbau des entwickelten Frameworks für die Inferenz-Partitionierung. Als Ergänzung zu den Vorstudien, ermöglicht CNN-

Parted die Evaluation von Systemen bestehend aus einer Kombination von analogen PIM-Komponenten aus neuartigen Speichertechnologien, wie z.B. RRAM oder MRAM, und digitalen Hardwarebeschleunigern. Dabei bezieht das Framework die unterschiedliche Rechenpräzision der Architekturen, d.h. die jeweilige Quantisierung von Aktivierungen und Gewichten, sowie die intrinsische Fehleranfälligkeit der PIM-Beschleuniger in die Suche nach einer Pareto-Front für die Inferenz-Partitionierung ein.

Als Eingabe verwendet CNNParted Dateien im ONNX-Format [157], welches viele DNN-Frameworks wie PyTorch oder TensorFlow unterstützen. Folglich ist eine hohe Interoperabilität und damit eine hohe Flexibilität gewährleistet, da Entwickler ihren gewohnten Toolflow verwenden und nicht auf spezielle Software umsteigen müssen. Mit der Eingabe des Netzes im ONNX-Format geht die interne Darstellung des Netzes als DAG einher, die notwendig ist, um im weiteren Verlauf die Verbindungen zwischen den einzelnen Schichten nachzuvollziehen. Für die Inferenz-Partitionierung wird aus dem Graphen, wie zuvor beschrieben, eine topologische Sortierung der Layer abgeleitet, um sicherzustellen, dass die Datenabhängigkeiten immer erfüllt sind. Basierend auf dieser geordneten Liste von Layern analysiert CNNParted im nächsten Schritt die Robustheit der einzelnen Schichten des DNNs. Dafür wird nach einer minimalen Layer-Quantisierung gesucht, welche anschließend Rückschlüsse darauf ziehen lässt, welche Schichten auf dem Beschleuniger mit der geringsten Quantisierung und Fehlerrate ausgeführt werden müssen. Genauere Informationen hierzu folgen in Abschnitt 5.4.3. Die Informationen dieser Analyse dienen in der Exploration als Einschränkung für das Mapping der Layer auf die vorhandenen Beschleunigerplattformen. Bevor jedoch mit diesem Teil begonnen werden kann, müssen die potenziellen Hardwarebeschleuniger zunächst im Hinblick auf ihre Latenz und ihren Energiebedarf pro Schicht sowie ihren Flächenbedarf untersucht werden. Dies geschieht, wie in Abschnitt 5.1 bereits ausführlich erläutert, anhand von Modellen der Beschleunigerarchitekturen, welche die Performance und Leistungsaufnahme pro Layer entsprechend abschätzen.

Die zentrale Komponente von CNNParted stellt der Explorationsalgorithmus dar, welcher dem in Definition 4 beschriebenen Verfahren entspricht und gemäß Definition 5 eine Pareto-Menge an vorteilhaften Partitionierungsschemata sucht. Dies geschieht basierend auf den zuvor ermittelten nichtfunktionalen Metriken und den Ergebnissen der Robustheitsanalyse. Hierbei wird durch die zuvor generierte topologische Sortierung sichergestellt, dass der Algorithmus alle Datenabhängigkeiten während der Exploration beachtet. Abschließend werden die Punkte der Pareto-Front im Hinblick auf die Genauigkeit des DNN analysiert, was schließlich zu einer Liste möglicher Pareto-

optimaler Partitionierungsschemata für das gegebene System führt. Diese Liste stellt gleichzeitig die Ausgabe von CNNParted dar.

5.4.1 Graphinitialisierung

Wie bereits erwähnt, stellt das ONNX-Format das gegebenen DNN bereits als gerichteten Graph dar. Folglich stehen die Informationen über die Hyperparameter sowie der Vorgänger und Nachfolger pro Layer bereits durch die Eingabe in CNNParted für die weitere Verwendung zur Verfügung. Allerdings liegt der Graph in einer derartigen Implementierung vor, dass diese inkompatibel zu existierenden Python-Frameworks zur Untersuchung von Graphen wie NetworkX [158] sind. Folglich extrahiert dieses CNNParted-Modul die Informationen aus der ONNX-Eingabe und überführt dies in eine, mit vielen anderen Tools kompatiblen Repräsentation des DAGs. Dazu gehören vor allem auch die Hyperparameter der CONV- und Fully-Connected-Layer, welche insbesondere für die weiteren Evaluationen relevant sind. Diese Repräsentation ermöglicht so die Analyse des Graphen hinsichtlich verschiedener Aspekte und ist somit ein wichtiger Bestandteil von CNNParted. Dadurch können neben der topologische Sortierung zukünftig weitere Graph-Operationen in das Framework integriert und für Analysen des DNN verwendet werden.

5.4.2 Topologische Sortierung

Gemäß der allgemeinen Problemdefinition in Abschnitt 5.3 ist eine topologische Sortierung des Layer-Graph notwendig, um eine Partitionierung an allen Stellen im DNN ohne den Einsatz komplexerer Algorithmen zu evaluieren. Die Bestimmung topologischer Sortierungen aus einem Graph kann dabei auf unterschiedlichen Wegen erfolgen. Allgemein lassen sich hierfür beispielsweise klassische Verfahren zur Lösung des Scheduling-Problems, wie As-Soon-As-Possible (ASAP) und As-Late-As-Possible (ALAP) heranziehen. Aufgrund des datenflusszentrierten Charakters von DNNs gibt es hier allerdings keine Layer, welche mit einer höheren Priorität als andere auszuführen sind. Lediglich die Einhaltung der Präzedenzrelationen muss gewährleistet sein, um sicherzustellen, dass die benötigten Eingabedaten eines Layers immer vorliegen. In der Regel kommen hierfür konstruktive Algorithmen zum Einsatz, welche das Scheduling Knoten um Knoten aufbauen. Verfahren wie ASAP und ALAP beginnen vom ersten bzw. letzten Knoten des DAGs und bilden von dort aus schrittweise die topologische Sortierung [159]. Beide Ansätze

lassen sich recht einfach, z.B. in Form des Hu-Algorithmus implementieren. Der Algorithmus nummeriert zunächst die Knoten durch, wobei dieses Label die Priorität für das Scheduling des Knotens festlegt. Im nächsten Schritt generiert das Verfahren einen Schedule, indem durchführbare Operationen entsprechend ihrer Priorität eingereiht werden. Nimmt man den einfachsten Fall des Schedulings an, in welchem lediglich eine Einheit zur Ausführung der Operationen zur Verfügung steht, ergibt sich folglich eine topologische Sortierung. Einen ähnlichen Ansatz verfolgt Kahns Algorithmus [160], welcher die Zahl der Vorgängerknoten als Entscheidungsgrundlage nimmt. Dieser wird auch von NetworkX [158] für die topologische Sortierung von DAGs verwendet, sodass auch CNNParted darauf zurückgreifen kann. Der Algorithmus durchsucht eine Liste der noch nicht einsortierten Knoten nach solchen ohne Vorgänger. Diese werden anschließend einsortiert und aus dem Graph gelöscht und die Liste, welche die Anzahl der Vorgängerknoten enthält, vor dem nächsten Durchlauf aktualisiert. Ähnlich zu ASAP und ALAP hängt die Einordnung der Knoten für den Fall, dass mehrere Kandidaten existieren, von der Implementierung ab. Folglich existieren verschiedene Varianten der beiden Algorithmen.

Bei der Suche nach einem vorteilhaften Partitionierungsschema hat auch die topologische Sortierung einen Einfluss, insbesondere in Bezug auf die benötigten Linkbandbreite an den Partitionierungspunkten. Folglich ist es sinnvoll, verschiedene Sortierungen hinsichtlich der Inferenz-Partitionierung zu untersuchen. Dies lässt sich durch Einbindung von Zufallsvariablen realisieren. Ein solches Verfahren wurde von Kamath et al. [93] vorgeschlagen und schließlich ebenfalls in CNNParted integriert. Der präsentierte Uniform Sampling Algorithmus basiert auf Kahns Algorithmus und nutzt einen Zufallsmechanismus zur Auswahl des Knotens. Dabei geht die Suche allerdings nicht vom Start- oder Endknoten des Graphen aus, sondern einem zufälligen Knoten im DAG. Von dort wird die topologische Sortierung gleichzeitig in beide Richtungen aufgebaut. Vorteil dieses Ansatzes ist, dass der Suchraum besser exploriert und damit mehr potenzielle Lösungen gefunden werden. Folglich bezieht die Suche nach einem optimierten Partitionierungsschema mehrere topologische Sortierungen ein und macht damit bessere Aufteilungen des Netzes über die vorhandenen Beschleunigerplattformen ausfindig.

5.4.3 Robustheitsanalyse

Die Einbindung einer funktionalen Metrik in die Exploration ist wichtig, um Lösungen direkt auszuschließen und dementsprechend nicht weiter zu ver-

folgen. Wie zuvor erwähnt, handelt es sich im Kontext von DNNs hierbei um die Genauigkeit als wesentliche funktionale Metrik. Allerdings dauert die Evaluation je nach System und DNN-Architektur mehrere Sekunden für rein digitale Systeme, im Falle der Inferenz auf PIM-Komponenten erhöht sich die Simulationszeit zusätzlich. Grund dafür ist die notwendige Betrachtung analoger Charakteristika, wie beispielsweise den Widerstand jeder einzelnen RRAM-Zelle, welche zusätzliche Berechnungen erfordern. So ergibt sich für ein ResNet-18 bei vollständiger Inferenz in PIM-Modulen und Simulation mittels des Open-Source Frameworks MNSIM [161] eine Evaluationsdauer von 37,3 s auf einer GPU. Für die Exploration der Inferenz-Partitionierung müssen in der Regel mehr als 10 000 valide Lösungen evaluiert werden, woraus sich ein Rechenaufwand von über 103,6 GPU Stunden ergibt [162]. In Konsequenz lässt sich festhalten, dass eine Evaluierung der Genauigkeit des DNNs im Rahmen der Suche nach validen Partitionierungsschemata mit einem solchen Ansatz nicht realisierbar ist.

Um dennoch diese wichtige funktionale Metrik indirekt einzubeziehen, nutzt CNNParted die Robustheitsanalyse im Vorlauf der Exploration und verwendet deren Ergebnisse bei der Suche nach einem vorteilhaften Partitionierungsschema in Form von Einschränkungen des Mappings der Partitionen auf Beschleunigerplattformen. Für die Identifikation derjenigen Schichten des DNNs, die von einer stromsparenden, wenngleich ungenauen PIM-basierten Berechnung und welche von einer präziseren, jedoch energieintensiveren digitalen Verarbeitung profitiert, ist eine entsprechende Analyse des Netzes unabdingbar. Nach dem derzeitigen Stand der Technik wird die Robustheit der Schichten meist durch Berechnung der Empfindlichkeit der einzelnen Parameter ermittelt [85]. Allerdings ist zu berücksichtigen, dass die Analyse auf zwei NVIDIA GTX1080ti-GPUs für ein ResNet-18 etwa eine Stunde pro Eigenvektor in Anspruch nimmt. Im Rahmen der Robustheitsanalyse ist eine exakte Bestimmung der Sensitivität jedoch nicht erforderlich. Das Ziel besteht vielmehr darin, den Designraum zu reduzieren, um nachteilige Partitionierungsschemata frühzeitig auszusortieren. Als Grundlage dafür dient daher die Mixed-Precision-Analyse des DNNs.

Die Robustheit der einzelnen Schichten gegenüber Quantisierung, welche recht einfach zu bestimmen ist [Hot23c], stellt einen guten Indikator für diese Entscheidung dar. In diesem Schritt untersucht CNNParted folglich die Auswirkungen der Fake-Quantisierung mit gemischter Bit-Präzision auf die Modellgenauigkeit. In Anbetracht der Tatsache, dass der Suchraum, bestehend aus allen möglichen Kombinationen der Layer-Quantisierung, im vorliegenden Fall relativ groß wäre, werden verschiedene Einschränkungen eingesetzt, um die Anzahl der zu möglichen Konfigurationen zu reduzieren.

Hinsichtlich der Quantisierung von Gewichten und Aktivierungen wird zunächst die Annahme aufgestellt, dass diese identisch ist. In der Konsequenz reduziert sich die Anzahl der Variablen für die Optimierung um die Hälfte. Andererseits erlaubt CNNParted bei der Exploration lediglich die Berücksichtigung relevanter Bitbreiten, welche in der vorhandenen Hardware des Systems genutzt werden. In der Konsequenz führt dieser Ansatz zu einer signifikanten Reduzierung des Suchraums, was eine wesentliche Verkürzung der Exploration zur Folge hat.

Im Gegensatz zu dem von Hotfilter et al. [Hot23c] vorgeschlagenen Ansatz, der hierfür einen GA vorsieht, basiert diese Analyse in CNNParted auf dem heuristischen Simulated Annealing Verfahren. Dies hat den Vorteil, dass der Algorithmus über weniger Parameter verfügt, welche entsprechend des Optimierungsproblems angepasst werden müssen. Für die Implementierung kommt die Variante des Verfahrens aus der SciPy-Bibliothek [163] zum Einsatz, welche zur Laufzeitoptimierung auf zehn globale Suchiterationen beschränkt wird. Ziel des Verfahrens ist es, eine Konfiguration des DNNs zu finden, welche die Layer jeweils möglichst stark quantisiert und dabei gleichzeitig eine hohe Genauigkeit erzielt. Aus diesen Anforderungen lässt sich folgende Kostenfunktion c ableiten:

$$c = \sum_{i=1}^{|Q|} b_i \cdot |acc - (ref - \delta)|\,, \tag{5.3}$$

wobei Q der Menge aller quantisierten CONV- und Fully-Connected-Layer sowie deren jeweilige Bitbreite b_i, acc der ermittelten Top-1-Genauigkeit dieser Konfiguration, ref der Referenz-Top-1-Genauigkeit und δ einem benutzerdefinierten erlaubten Verlust der Genauigkeit entspricht. Als Referenz ref dient die Genauigkeit des Netzes mit der geringsten Quantisierung, d.h. 16-bit-Integer-Quantisierung aller Layer. Da davon auszugehen ist, dass eine stärkere Quantisierung zu einer Verringerung der Genauigkeit führt, wird mittels δ eine relative Abweichung von ref angegeben, auf welche die Optimierung die Quantisierung hin anpasst. In Kombination mit der vorangestellten Summe sucht dieses Verfahren folglich nach einer optimalen Konfiguration des quantisierten DNN hinsichtlich minimaler Bitbreiten pro Layer und gleichzeitig maximal möglicher Genauigkeit.

Das Ergebnis der Robustheitsanalyse ist somit eine Zuordnung der maximalen Quantisierung pro Layer. Im Falle digitaler Beschleuniger, bei denen sich die Bitbreiten aus der Architektur ablesen lassen, wird somit direkt ausgewertet, welche auf dieser Plattform ausgeführt werden dürfen und welche nicht. Ergibt sich für ein Layer eine maximale Quantisierung von 16-Bit, so wird die

Inferenz auf einem 8-Bit-Beschleuniger in der anschließenden Exploration untersagt. Konträr dazu ergibt sich die Bit-Präzision analoger PIM-Beschleuniger anhand der internen Crossbar sowie der Präzision der integrierten ADCs. Dabei ist zu beachten, dass PIM-Komponenten eine recht hohe statische Fehlerrate durch defekte Speicherzellen aufweisen. Folglich muss hier eine zusätzliche Kompensation erfolgen, indem die Anforderungen der Layer an die Präzision des PIM-Moduls erhöht werden. So würde beispielsweise eine Verschiebung der Einschränkung um 2 Bits dafür sorgen, dass ein PIM-Beschleuniger mit realer Präzision von 8 Bits ausschließlich Layer mit einer Quantisierung von weniger als 6 Bits ausführen darf.

5.4.4 Beschleuniger-Modellierung

Neben der Robustheitsanalyse führt CNNParted weitere Untersuchungen im Vorfeld durch, um die mehrfache Ausführung gleicher Evaluationen zu verhindern und somit die Gesamtlaufzeit der Simulationen signifikant zu senken. Dabei handelt es sich um die Analyse von Latenz und Energiebedarf eines jeden Layers auf jedem verfügbaren Hardwarebeschleuniger des Systems, sowie eine Abschätzung des Flächenbedarfs der Beschleunigerplattformen. Folglich greift der Explorationsalgorithmus für die Bewertung eines Partitionierungsschemas direkt auf die vorliegenden Ergebnisse zurück, wodurch zusätzliche Berechnungen entfallen. Hierbei muss allerdings in der Simulation explizit zwischen analogen In-Memory und digitalen Beschleunigern differenziert werden, da diese sich grundlegend in ihrem Aufbau und ihren Eigenschaften unterscheiden.

Digitale Beschleuniger

Wie bereits in Abschnitt 5.1 beschrieben, eignet sich das Open-Source Tool Timeloop [139] im Zusammenspiel mit Accelergy [141] sehr gut für die Evaluation digitaler Beschleuniger und kommt daher in CNNParted zum Einsatz. Im Gegensatz zu zyklenakkuraten Simulationsumgebung bieten analytische Modelle von Hardwarebeschleunigern zwar lediglich Abschätzungen und keine exakten Werte relevanter Metriken, sind dabei aber signifikant schneller [Hot23a]. Da CNNParted für den Einsatz zur Entwurfszeit konzipiert ist, zielt es auf eine schnelle Exploration ab, um potenziell vorteilhafte Partitionierungsschemata zu finden und bei der Optimierung der Systemarchitektur zu unterstützen. Eine exakte Bestimmung der Werte ist nicht erforderlich, weshalb CNNParted Timeloop für die Evaluierung digitaler Beschleuniger

integriert. Es nutzt abstrakte Modelle der Hardwarearchitektur, um die nichtfunktionalen Metriken Latenz und Energiebedarf der einzelnen Layer abzuschätzen. Dabei kann das Tool auch externen Speicher einbeziehen, wie beispielsweise Double Data Rate (DDR)-Random-Access Memory (RAM), welcher im Gegensatz zu den verfügbaren Speicherblöcken in Hardwarebeschleunigern ausreichend groß für die Parameter eines DNNs ist. Somit findet auch die benötigte Datenübertragung zwischen den genannten Komponenten Berücksichtigung hinsichtlich Latenz und Energiebedarf. Timeloop sucht nach einem optimalen Mapping der Layer auf den Beschleuniger hinsichtlich eines benutzerdefinierten Optimierungsziels. Für die Inferenz-Partitionierung ist in der Regel eine gute Balance zwischen den beiden Metriken sinnvoll, daher optimiert Timeloop in den nachfolgenden Anwendungsstudien von CNNParted das Mapping hinsichtlich dem Energy-Delay-Produkt (EDP). Zusätzlich erlaubt die Erweiterung von Timeloop um Accelergy auch eine Abschätzung des Flächenbedarfs abhängig vom verwendeten Technologieknoten. Folglich liefert diese Toolchain alle relevanten Metriken der digitalen Beschleuniger für die Exploration valider Partitionierungsschemata.

Analoge Processing-In-Memory Beschleuniger

Aufgrund der hohen Bandbreitenanforderungen von DNNs an die Hardware, stehen analoge PIM-Beschleuniger aktuell sehr im Fokus, wie bereits in Abschnitt 2.2 erläutert. So finden sich im aktuellen Stand der Technik verschiedene Realisierungen solcher Systeme wieder, hauptsächlich unter Verwendung von RRAM und MRAM. Auch Ansätze, welche Multi-Bit Zellen einsetzen, um die Speicherdichte zu erhöhen, lassen sich in einigen wissenschaftlichen Publikationen finden. Allerdings leiden diese unter unausgereiften Prozesstechnologie sowie äußeren Einflüssen und damit einer höheren Fehlerrate im Vergleich zu Single-Bit Zellen [164, 165, 166]. Gleichzeitig erfordern insbesondere Architekturen mit mehr als 2 Bits pro Zelle deutlich komplexere und damit weniger effiziente ADCs, welche letztlich den Vorteil der erhöhten Speicherdichte zunichtemachen [167]. Folglich unterstützt die hier beschriebene Implementierung von CNNParted lediglich Single-Bit Zellen in PIM-Beschleunigern. Als Grundlage dient die Simulationsumgebung MNSIM [161], welche eine hierarchische Modellierung der Architekturen und dabei auch die Untersuchung verschiedener Speichertechnologien erlaubt.

Digitale Beschleuniger nutzen häufig eine optimierte Speicherhierarchie mit Caches, um eine hohe Performance zu erzielen. Dabei ist es nicht notwendig, alle Gewichte im Beschleuniger selbst abzulegen, ein breitbandiger Anschluss

an den Hauptspeicher ist hierfür in der Regel ausreichend. Folglich hat die Anzahl der Parameter eines Netzes keine Auswirkung auf den Flächenbedarf des digitalen Beschleunigers. Im Gegensatz dazu ist dies im Falle von PIM-Beschleunigern, welche auf nicht-flüchtigen Speichertechnologien wie RRAM und MRAM basieren, nicht möglich. Grund dafür ist die geringe Anzahl an möglichen Schreibzyklen sowie eine erhöhte Schreibdauer und -energie im Vergleich zu Static Random-Access Memory (SRAM) [168, 169]. Demnach müssen die Parameter des DNNs dauerhaft im PIM-Beschleuniger vorliegen, wodurch der Flächenbedarf signifikant von der Anzahl an Gewichten abhängt. Entsprechend schätzt MNSIM diese Metrik für die weiteren Evaluationen pro Layer ab, sodass insbesondere dieser Wert bei der Suche nach einem optimierten Partitionierungsschema Beachtung findet. Für den Energiebedarf analysiert die Simulationsumgebung die verschiedenen Komponenten des Beschleunigers, wobei insbesondere die ADCs einen gewichtigen Einfluss auf die Leistungsaufnahme haben.

Aufgrund des Aufbaus solcher PIM-Beschleuniger kann die Inferenz recht einfach gepipelined ausgeführt und somit ein deutlich höherer Datendurchsatz erzielt werden [49, 50]. Folglich geht MNSIM in der Modellierung des Beschleunigers davon aus, dass die Ausführung eines DNNs derart realisiert ist. Die Latenz d_{total} für die gesamte Inferenz im PIM-Beschleuniger ist somit gegeben durch:

$$d_{f,l} = d_{f,l_{pre}} + t_{trans} + t_{comp} \tag{5.4}$$

$$d_{total} = d_{f,l_{end}} \tag{5.5}$$

wobei $d_{f,l}$ die Zeit angibt, zu der eine Recheneinheit f des Beschleunigers die Verarbeitung eines bestimmten Layers l abgeschlossen hat. Diese ergibt sich aus der Ausführungszeit der vorherigen Schichten $d_{f,l_{pre}}$ sowie der Übertragungsdauer der Aktivierungen t_{trans} und der Berechnungsdauer t_{comp} des Layers. Für die Exploration der Inferenz-Partitionierung ist es allerdings erforderlich, die Latenzen für jede DNN-Schicht zu ermitteln. Basierend auf der Gleichung (5.4) ergibt sich folglich eine Abschätzung der Latenz eines einzelnen Layers d_{layer} durch:

$$d_{layer} \approx max\left(d_{f,l}\right) - max\left(d_{f,l_{pre}}\right) \tag{5.6}$$

wobei $max\left(d_{f,l}\right)$ den Zeitpunkt markiert, an dem alle Einheiten des Beschleunigers die Verarbeitung des Layers l abgeschlossen haben.

5.4.5 Link-Modellierung

Neben den Beschleunigerplattformen im System ist darüber hinaus die Modellierung der Links zwischen ebenjenen Hardwaremodulen für eine vollständige Simulation notwendig. Konzeptionell ist dies die Verbindung, die Aktivierungen zwischen zwei benachbarten Schichten des DNNs überträgt. Im Gegensatz zur Evaluation der Beschleuniger, wird in diesem Fall allerdings auf einfache mathematische Modelle zurückgegriffen, welche eine sehr geringe Rechenzeit beanspruchen. Folglich ist es möglich, die relevanten Metriken des Links für jedes valide Partitionierungsschema innerhalb der Exploration zu ermitteln. Um die Auswirkungen des Links die Inferenz des DNNs abzuschätzen, werden der Energiebedarf des Links und die Übertragungslatenz berücksichtigt

Im Rahmen der vorliegenden Arbeit wurden in den nachfolgenden Anwendungsstudien zwei unterschiedliche eingebettete Systeme untersucht. Einerseits verteilte eingebettete Systeme, welche, wie bereits in Abschnitt 5.1 erwähnt, via Ethernet miteinander verbunden sind. Diese sind beispielsweise im Automobil zu finden, wobei die Daten zwischen Sensor und dem Steuergerät, der ECU, weitere Rechenplattformen durchlaufen. Zum anderen werden Chiplet-basierte Systeme betrachtet, die aktuell im Zentrum der Forschung stehen. Solche Plattformen kombinieren Chiplets über einen sogenannten Interposer, welcher für eine breitbandige und schnelle Verbindung zwischen den Modulen sorgen. Als Beispiel für einen solchen Link betrachtet diese Arbeit UCIe.

Ethernet

Verteilte eingebettete Systeme stellen speziell in Anwendungen wie dem autonomen Fahren hohe Anforderungen an die benötigte Link-Bandbreite. Gleichzeitig müssen geringe Latenzen sowie Zuverlässigkeit und Vorhersagbarkeit für einen Einsatz in solchen sicherheitskritischen Aufgaben gewährleistet sein. Im Automobilbereich galt viele Jahre der Controller Area Network (CAN)-Bus als wichtigster Standard, allerdings ist dessen Bandbreite stark limitiert und damit für zukünftige Implementierungen von ADAS eher ungeeignet [170, 171]. Durch Entwicklungen wie Time-Sensitive Networking (TSN) hat sich in den letzten Jahren Ethernet hervorgetan, welches durch diese Erweiterungen die Anforderungen sicherheitskritischer Anwendungen erfüllt [172, 173]. Neben solchen Anwendungen findet sich Ethernet heutzutage aber auch in vie-

len Bereichen des IoT wieder, weshalb dessen Charakteristika als Link-Modell für verteilte eingebettete Systeme dienen soll.

CNNParted integriert dabei das in Abschnitt 5.1 beschriebene Ethernet-Modell, welches die für die Abschätzung des Energiebedarfs die Leistungsaufnahme typischer PHYs verwendet und die Latenz auf Grundlage von Übertragungsrate und Signallaufzeit in Kupfer ermittelt. Die Berechnung der Latenz basiert dabei auf einer deterministischen Datenübertragung, d.h. der Link überträgt eine bestimmte Datenmenge in gleichen Zeitabständen. Das Modell der Übertragungslatenz für Ethernet hängt damit also von vier benutzerdefinierten Parametern ab: Gesamtdatenmenge, die vom DNN-Layer erzeugt und übertragen wird, maximale Ethernet-Paketgröße, Verbindungsbandbreite und Kabellänge. Somit schätzt CNNParted die Latenz und den Energiebedarf des Links und der zugehörigen Komponenten für jedes potenzielle Partitionierungsschema in kurzer Zeit ab, weshalb dieser Ansatz eine bandbreitenorientierte Bewertung der Inferenz-Partitionierung zur Entwurfszeit innerhalb der Partitionierungs-Exploration ermöglicht.

UCIe

Die Hardware-Anforderungen von DNN-basierten Anwendungen wie ADAS oder Assistenzrobotern steigen ständig in Bezug auf Latenz, Datendurchsatz und maximalen Energiebedarf. Zur Bewältigung dieser Herausforderungen ist eine Vielzahl von hoch spezialisierten Beschleunigern erforderlich, um die geeignete Hardware für die verschiedenen Anwendungen bereitzustellen, die solche eingebetteten Plattformen ausführen müssen. Darüber hinaus erhöhen sicherheitsrelevante Hardwarekomponenten, die in diese Anwendungsfälle integriert werden müssen, die Komplexität des Systems weiter. Mit zunehmender Chipgröße sinkt die Ausbeute jedoch erheblich. In der Folge kommt es zu einem enormen Anstieg der Kosten [174]. Dagegen bieten Chiplet-basierte Architekturen niedrigere Kosten und erzielen eine höhere Energieeffizienz. Dies führte in den vergangenen Jahren zur Entwicklung eines ganzen Ökosystems, bestehend aus großen eingebetteten Systemen [175], Electronic Design Automation (EDA) Tools [176] und Chiplet Interconnects wie Universal Chiplet Interconnect Express (UCIe) [177].

Um in nachfolgenden Anwendungsstudien die Eignung von CNNParted zur Lösung der komplexen Aufgabe der Partitionierung von CNNs auf solchen

Systemen [178] nachzuweisen, umfasst die vorliegende Arbeit auch ein UCIe-Modell. Dabei handelt es sich um ein paralleles Interface mit geringem Energiebedarf. Ähnlich zur Modellierung von Ethernet ist in diesem Fall die Betrachtung der physikalischen Implementierung für eine Abschätzung relevanter Metriken nicht notwendig. Daher nutzt das UCIe-Modell in CNNParted für die Ermittlung von Latenz und Energiebedarf des Links die UCIe 1.0 Spezifikation gemäß [177].

5.4.6 Explorationsalgorithmus

Nachdem die Evaluierung der Hardwarebeschleuniger für die relevanten Schichten des DNN abgeschlossen ist, erfolgt im nächsten Schritt auf dieser Basis die Untersuchung valider Partitionierungsschemata. Dies verlangt die Generierung verschiedener gültiger Verteilungen der Arbeitslast auf die vorhandenen Hardwarebeschleuniger sowie eine Bewertung dieser hinsichtlich nicht-funktionaler Metriken. Die Einbeziehung der Genauigkeit des DNNs als relevante funktionale Metrik erfolgt, indem CNNParted die Ergebnisse der Robustheitsanalyse in der Exploration als Einschränkung verwendet. Dabei prüft das System iterativ für jede Schicht, ob die verwendete Bitbreite der Beschleunigerplattform, auf die sie gemappt werden soll, ausreichend ist. Dadurch werden bereits während der Exploration Partitionierungsschemata herausgefiltert, die mit hoher Wahrscheinlichkeit zu einer deutlichen Verringerung der Genauigkeit führen würden.

Für die Suche nach einer vorteilhaften Inferenz-Partitionierung des gegebenen DNNs, ist grundsätzlich der Einsatz unterschiedlicher Algorithmen denkbar. Allerdings scheidet ein exaktes Verfahren aufgrund der Vielzahl an möglichen Schemata aus. Soll ein kleines DNN bestehend aus 15 Layern und zwei Partitionierungspunkten für ein verteiltes eingebettetes System bestehend aus drei Plattformen optimiert werden, ergeben sich basierend auf der in Abschnitt 5.3 beschriebenen Problemdefinition

$$\binom{15+2-1}{2} = 120 \tag{5.7}$$

verschiedene Partitionierungsschemata. In diesem Fall wäre eine Evaluation aller Systemkonfigurationen noch gut möglich. Dies ändert sich allerdings, sobald etwas größere DNNs und komplexere Systemarchitekturen untersucht werden, welche die mehrfache Ausführung auf derselben Plattform erlauben. Konkret bedeutet dies für ein mittelgroßes DNN bestehend aus 100 Layern

und zehn maximal erlaubten Partitionierungspunkten sowie vier verfügbaren Beschleunigerplattformen eine Gesamtzahl von

$$\binom{100+10-1}{10} \cdot 4^{10+1} = 1,788 \cdot 10^{20} \tag{5.8}$$

theoretisch möglichen Partitionierungsschemata. Dieses Beispiel weist die Schwere des Problems und folglich die Notwendigkeit der Anwendung von Heuristiken nach. Aus diesem Grund kommt an dieser Stelle zur Exploration des Suchraums ein GA zum Einsatz. Im Gegensatz zu Methoden wie dem Reinforcement Learning (RL) bietet dieser den Vorteil, dass ein solcher Algorithmus bei geeigneter Wahl der Ausgangslösungen einen großen Suchraum abdeckt und keine Kostenfunktion definiert werden muss. Zudem existieren nur wenige Heuristiken, welche gut für die Optimierung von mehr als vier Metriken skalieren [179]. Der verwendete NSGA-II-Algorithmus ist ein solcher. Mithilfe von Crossover und Mutation exploriert dieser parallel verschiedene Partitionierungsschemata, um schließlich eine Pareto-Front auszugeben. Dabei versucht der iterative Optimierungsalgorithmus einen Eingangsvariablenvektor zu finden, der die einzelnen Zielfunktionen minimiert. Aufgrund der in Definition 4 geforderten Präzedenzrelation müssen die Partitionierungspunkte in dem Variablenvektor linear geordnet sein. Dies wird jedoch bei der Erstellung der Generationen vom Algorithmus nicht berücksichtigt und wird erst bei der Auswertung überprüft. Folglich nimmt die Generierung möglicher validen Lösungen mit dem GA viel Zeit in Anspruch, wenn die Ausgangslösung ungültig ist. Aus diesem Grund erhält der GA in CNNParted für eine schnelle Exploration ausschließlich valide Partitionierungsschemata als initiale Population.

Jedes Individuum in dieser Population hat ein Genom, das die Eingangsvariablen repräsentiert. Diese setzen sich in der Implementierung von CNNParted zusammen aus

- einem Vektor, welcher die Partitionierungspunkte gemäß Definition 3 festlegt, und
- einem Vektor, welcher das Mapping der Partitionen auf die verfügbaren Beschleunigerplattformen definiert.

Im nächsten Schritt evaluiert der GA die Individuen hinsichtlich der in Definition 5 genannten Metriken gemäß Pseudocode 2. Dies geschieht durch schrittweise Auswertung der einzelnen Layer entsprechend der verwendeten topologischen Sortierung. Dabei berücksichtigt CNNParted, wie nachfolgend

Pseudocode 2 Fitnessfunktion für die Evaluation der Partitionierungsschemata

```
1:  procedure EVALUATE_SCHEME(x)
2:      P ← get_scheme(x)              ▷ Partitionierungsschema aus Eingabe x ableiten
3:      S ← {}                         ▷ Initialisiere Liste der Nachfolger
4:      for p, acc in P do             ▷ Iteriere durch Partitionen
5:          if violate_bitwidth(p, acc) then
6:              return invalid partitioning   ▷ Abbruch bei Verletzung der Beschränkung
7:          end if
8:          L_l[p], E_l[p], B[p] ← eval_link(p, S)      ▷ Evaluiere Link-Parameter
9:          L_p[p], E_p[p] ← eval_partition(p)          ▷ Evaluiere Partitions-Parameter
10:         S ← get_successors(p)          ▷ Update Liste der Nachfolger
11:     end for
12:     latency ← sum(L_l, L_p)            ▷ Berechne Gesamtlatzenz
13:     energy ← sum(E_l, E_p)             ▷ Berechne Gesamtenergiebedarf
14:     th ← calc_throughput(L_l, L_p)     ▷ Berechne Durchsatz
15:     a ← get_area(P)                    ▷ Berechne Flächenbedarf
16:     return latency, energy, th, a, B   ▷ Rückmeldung der Bewertungsmetriken
17: end procedure
```

genauer erläutert, auch den Speicherbedarf einer Partition, um Partitionierungsschemata herauszufiltern, welche auf mindestens einer Beschleunigerplattform mehr Parameter als möglich speichern würden. Zentrale Recheneinheiten wie ECUs verfügen üblicherweise über genügend externen Speicher, sodass dieser Fall eher selten eintrifft und daher in der Exploration nicht Beachtung finden muss. Dagegen nutzen sensornahe Plattformen in der Regel nur den internen Speicher, welcher einen signifikanten Teil des Flächenbedarfs eines solchen SoCs ausmacht. Somit wird eine Evaluation der Speicheranforderungen als Teil der Suche nach einem validen Partitionierungsschema benötigt. Für die Inferenz eines topologisch sortierten DNNs von Layer l_n bis l_m auf einer Beschleunigerplattform a ist der Speicherbedarf gegeben durch:

$$m_a(l_n, l_m) = \left(\sum_{i=n}^{m} s_i + \max(f_{all,n}, \ldots, f_{all,m}) \right) \cdot b_a, \quad n \leq m \tag{5.9}$$

$$\text{mit} \quad f_{all,j} = f_{j,in} + f_{j,out}, \tag{5.10}$$

wobei s_i der Anzahl der Parameter von Layer l_i, $f_{all,j}$ der Summe der Größe der Eingangs- und Ausgangsaktivierungen $f_{j,in}$ und $f_{j,out}$ von Layer j, sowie b_a der Bitbreite der quantisierten Parameter und Aktivierungen entspricht. Auf dieser Grundlage berechnet CNNParted den Speicherbedarf jedes einzelnen Layers und entscheidet, ob die Anforderungen von den entsprechenden Plattformen erfüllt werden.

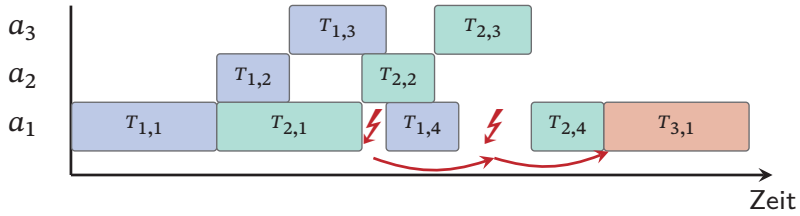


Abbildung 5.12: Schedulingproblem bei wiederholter Ausführung eines DNNs auf mehreren Plattformen $a_i \in A$

Um den Übertragungsaufwand des Links zu ermitteln, ist eine Analyse der zwischen den Partitionen zu übertragenden Daten notwendig. Dazu werden in jedem Schritt die Nachfolgeknoten des aktuellen Layers markiert. Sobald die Evaluation einen dieser Knoten erreicht, wird dessen Markierung entfernt. Ist das Ende einer Partition und damit ein Partitionierungspunkt erreicht, lässt sich somit anhand der noch markierten Layer berechnen, wie viele Daten zur nächsten Partition übertragen werden müssen.

Abschließend ermittelt der Evaluator anhand der Partitionierung und der Latenzen der einzelnen Partitionen zudem den möglichen Datendurchsatz. Diese Metrik ist für viele Anwendungen, wie z.B. auch dem autonomen Fahren, von großer Bedeutung, um sinnvolle Entscheidungen zu treffen. So kann die Ausführung der DNN-Inferenz auf einem einzigen Hardwarebeschleuniger gegebenenfalls zwar eine geringe Latenz bieten, dafür erreicht ein solcher Ansatz meist keinen hohen Datendurchsatz. Um diese Abwägung in die Suche nach einem vorteilhaften Partitionierungsschema für den Anwendungsfall einzubinden, muss diese Metrik folglich ebenfalls abgeschätzt werden. Im einfachsten Fall arbeiten die Plattformen als eine Pipeline, wobei jeder Beschleuniger maximal eine Partition bestehend aus zusammenhängenden Layern des DNNs verarbeitet. Somit ist der Datendurchsatz für ein beispielhaftes System bestehend aus den über einen Link verbundenen Plattformen a_1 und a_2 gegeben durch:

$$th(\vec{x}) = min\left(\frac{1}{d_{a_1}}, \frac{1}{d_{Link}}, \frac{1}{d_{a_2}}\right), \tag{5.11}$$

In Multi-Beschleuniger SoCs oder Chiplet-basierten Systemen ist die Berechnung des Datendurchsatzes dagegen nicht ganz so trivial. Der Grund dafür ist in Abbildung 5.12 dargestellt. Beispielhaft zeigt dies ein Partitionierungsschema für ein System bestehend aus drei Beschleunigerplattformen, welche

über die Zeit mehrfach die Inferenz des gegebenen DNNs ausführen sollen. Die Tasks $T_1 \dots T_3$ beschreiben dabei jeweils eine Iteration. Abbildung 5.12 veranschaulicht, dass bei Ausführung der dritten Wiederholung ein Problem entsteht, da Plattform a_1 für die Inferenz der ersten Partition $T_{3,1}$ erst nach Abschluss der vierten Partition der zweiten Iteration $T_{2,4}$ zur Verfügung steht. Somit kann an dieser Stelle nicht auf Gleichung (5.11) für die Berechnung des Datendurchsatz zurückgegriffen werden. CNNParted geht in seiner Abschätzung in einem solchen Fall von einem Worst-Case-Szenario aus. Genauer analysiert CNNParted für jede Plattform $a \in A$ die Zeit zwischen Beginn $t_{i,start}$ der ersten auf ihr ausgeführten Partition i und dem Ende $t_{j,end}$ der letzten auf ihr ausgeführten Partition j und definiert dies als Latenz d_a des Beschleunigers für die Berechnung des Datendurchsatzes:

$$d_a = t_{j,end} - t_{i,start} \tag{5.12}$$

Für das Beispiel in Abbildung 5.12 bedeutet dies, dass der Datendurchsatz $th(\vec{x})$ gemäß Gleichung (5.11) durch Beschleuniger a_1 definiert wird. Dabei entspricht die Gesamtlatenz dieser Plattform für die Ausführung der Inferenz $d_{a_1} = t_{4,end} - t_{1,start}$, wobei $t_{1,start}$ den Start- der ersten Partition und $t_{4,end}$ den Endzeitpunkt der vierten Partition markiert. Für den Datendurchsatz des Partitionierungsschemas $\vec{x}$ ergibt sich in diesem Fall somit $th(\vec{x}) = \frac{1}{d_{a_1}}$.

Sobald eine Bewertung aller Individuen einer Generation erfolgt ist, werden diese jeweils mit einer Fitnessfunktion bestehend aus den Zielfunktionen und dem Grad der Verletzungen der Einschränkungen versehen. Auf dieser Grundlage wählt der GA anschließend die besten Partitionierungsschemata sowie ein paar zufällig ausgewählte Individuen aus, welche die Basis der nächsten Generation bilden. Sie stellen dabei die Eltern dar, d.h. der nachfolgende Crossover-Schritt verschränkt die Genome von zwei oder mehr Elternteilen, wobei die Variablen jeweils zufällig aus einem dieser Individuen gewählt werden. Bevor die Population der nächsten Generation erstellt ist, erfolgt zudem noch eine randomisierte Mutation der Genome. Alle diese Schritte inklusive der Evaluation der Individuen wird für jede Generation wiederholt, bis der GA ein vorher definiertes Abbruchkriterium erfüllt. Durch diesen Ansatz kann der GA eine Konvergenz hin zu Pareto-optimalen Partitionierungsschemata gewährleisten.

Für die Exploration valider Partitionierungsschemata und Generierung der Pareto-Front kommt in CNNParted der NSGA-II [59] zum Einsatz, welcher auch in der Python-Bibliothek Pymoo [180] zur Verfügung steht. Dieser Al-

gorithmus nutzt zusätzliche Strategien im Vergleich zu anderen GAs, welche für eine bessere Performance bei Mehrzieloptimierungen sorgen. Dazu zählt insbesondere, dass er eine Menge an Pareto-optimaler Lösungen ohne Crossover und Mutation in die nächste Generation übernimmt, womit auch die Rechenkomplexität sinkt. In CNNParted bricht die Exploration ab, sobald der GA eine bestimmte Zahl an Generation erreicht. Dieser Wert richtet sich dabei an der Größe des DNNs, da sicher der Suchraum mit steigender Anzahl an Layern vergrößert. Ähnlich dazu verhält es sich mit der Populationsgröße, welche ebenfalls entsprechend der Größe des DNNs gewählt wird. Diese beiden Parameter definieren somit die potenzielle Anzahl an evaluierbaren Eingangsvariablenvektoren und folglich auch die Laufzeit des Algorithmus sowie die Wahrscheinlichkeit, alle Pareto-optimalen Punkte des gegebenen Partitionierungsproblems zu finden.

5.4.7 Funktionale Evaluation

Mittels der Robustheitsanalyse filtert CNNParted bereits valide Partitionierungsschemata heraus, welche mit großer Wahrscheinlichkeit einen zu großen negativen Einfluss auf die Genauigkeit des DNNs haben. Allerdings erfolgt hierbei lediglich eine sehr grobe Abschätzung der Metrik, die für die Suche nach einer vorteilhaften Partitionierung des Netzes nicht ausreichend ist. Somit ist eine funktionale Evaluation vonnöten, um präzisere Aussagen über die unterschiedlichen Auswirkungen der einzelnen Partitionierungsschemata auf die Genauigkeit zu treffen. Dieser finale Schritt ist, wie bereits in Abschnitt 5.4.3 erwähnt, vergleichsweise zeitaufwendig, weshalb CNNParted dabei nur die Pareto-optimalen Partitionierungen auf ihre Genauigkeit untersucht.

Wie bereits im Falle der Robustheitsanalyse dargestellt, sind hierfür die Gewichte und Aktivierungen des DNNs zur Ausführung auf den Beschleunigern entsprechend deren Hardwareparameter zu quantisieren. Für die funktionale Evaluation muss zusätzlich die Quantisierung durch den ADC in den analogen PIM-Beschleunigern betrachtet werden, welche ebenfalls zu Berechnungsfehlern führt. Darüber hinaus ist es für eine realistische Schätzung der Genauigkeit unerlässlich, neben dem Quantisierungsfehler auch die Stuck-At Faults (SAFs) in PIM-Architekturen zu berücksichtigen. Dabei handelt es sich um statische Fehler der Speicherzellen, welche dauerhaft in einem der beiden binären Zustände verbleiben. Diese SAFs lassen sich zwar durch einen Built-in self-test (BIST) erkennen, allerdings ist die Fehlerrate in solchen PIM-Modulen mit meist mehr als 10 % [181] derart hoch, dass eine große

räumliche Redundanz für eine fehlerfreie Durchführung der Inferenz notwendig wäre. Statische Fehler können durch unterschiedliche Effekte zustande kommen [182] und sind daher schwer zu modellieren. Viele Verfahren nehmen als Vereinfachung eine diskrete Gleichverteilung der SAFs über die Zellen und damit die Bits der Gewichte an [183, 184, 185].

Um SAFs zu modellieren, führt CNNParted während der Inferenz Bit-Invertierungen in den Gewichten durch, abhängig von der SAF-Rate der analogen Beschleuniger. Konkret betrachtet CNNParted hierfür sowohl CONV- als auch Fully-Connected Layer, um die Auswirkungen von SAFs auf die Genauigkeit zu untersuchen. Dafür wählt das Framework zufällig n Bits der Gewichte dieser Schichten gleich verteilt über das gesamte DNN aus. Die Zahl n hängt dabei von der SAF-Rate des Beschleunigers ab. Besteht das DNN beispielsweise aus 1000 8 − Bit Gewichten, so werden bei einer realen Fehlerrate von 10 % insgesamt 800 Bits manipuliert und damit invertiert. Dabei ist zu beachten, dass die SAF-Rate nicht automatisch der realen Fehlerrate entspricht, da insbesondere im Falle von SAF-0 die Wahrscheinlichkeit geringer ausfällt, dass ein Bit in der realen Hardware falsch wäre. Grund dafür ist die hohe Anzahl an Gewichten mit einem Wert nahe null [186, 187]. Folglich ist die reale Fehlerrate vor allem durch die SAF-1-Rate bestimmt. Die Implementierung in CNNParted erlaubt, eine Fehlerrate für alle verfügbaren Hardwareplattformen und damit auch für digitale Hardwarebeschleuniger zu definieren. Auf diese Weise lassen sich mittels derselben Methodik zusätzlich auch Einflüsse von Low-Power und Undervolting-Szenarien untersuchen. Abgesehen von der Inferenz mit Testdaten ermöglicht das Framework optional auch die Durchführung von QAT. Dies ist insbesondere dann sinnvoll, wenn die DNN-Architektur insgesamt wenig robust gegenüber Quantisierung von Gewichten und Aktivierungen ist. Die Quantisierung der einzelnen Layer erfolgt in Abhängigkeit der jeweiligen Beschleunigerplattform und wird durch das Partitionierungsschema definiert. Daher führt CNNParted das Training entsprechend für jede Pareto-optimale Lösung durch und evaluiert anschließend die Genauigkeit der Inferenz.

5.5 Anwendungsstudien

Dieser Abschnitt untersucht die Anwendung von CNNParted für zwei beispielhafte Systeme. Wie bereits zuvor erwähnt, liegt der Fokus dabei auf verteilten eingebetteten und Chiplet-basierten Systemen. Im erstgenannten Anwendungsbeispiel kommen ausschließlich digitale Beschleuniger zum Einsatz,

Abbildung 5.13: Übersicht der Systemarchitektur der ersten Anwendungsstudie

welche via Ethernet verbunden sind. Dies soll ein typisches dezentrales System darstellen und damit Rückschlüsse auf die Auswirkungen der Inferenz-Partitionierung auf existierende Architekturen ermöglichen. Dagegen untersucht die Anwendungsstudie der Chiplet-basierten Systeme zusätzlich auch den Einfluss analoger PIM-Beschleuniger auf die Inferenz-Partitionierung. Aufgrund der hohen Bandbreite und geringen Latenz des Links ist es in diesem Fall möglich, einen Hardwarebeschleuniger mehrfach für die Inferenz zu nutzen, d.h. mehrere Partitionen auszuführen.

5.5.1 Verteilte eingebettete Systeme

Zunächst sollen die Auswirkungen der Inferenz-Partitionierung auf ein verteiltes eingebettetes System unter Beachtung der hierarchischen Anordnung der Plattformen untersucht werden. Die nachfolgende Evaluation geht hierfür beispielhaft von einer Konfiguration bestehend aus drei Beschleunigern aus, welche sequenziell die entsprechenden Layer ausführen. Somit ergibt sich als Randbedingung, dass Daten lediglich von der ersten über die zweite hin zur dritten Plattform übertragen werden dürfen. Dies impliziert, dass jeder Beschleuniger lediglich eine Partition ausführen darf, welche aus zusammenhängenden Layern besteht. Eine schematische Übersicht des untersuchten Systems ist in Abbildung 5.13 gegeben.

Als Link fungiert in diesem Fall Gigabit-Ethernet mit einer Kabellänge von jeweils fünf Metern. Die drei verwendeten Plattformen basieren auf der 16-Bit Gemmini- [41], 16-Bit Eyeriss- [47] und 8-Bit Simba-Architektur [40], welche jeweils mit einer Frequenz von 200 MHz getaktet sind. Für die Evaluation dieser Beschleuniger hinsichtlich Latenz und Energie, sucht Timeloop nach einem Mapping der Layer auf die Architektur, welches ein bestmögliches EDP liefert. Bei den verwendeten DNNs handelt es sich um CNNs zur Bildklassifizierung, welche Bilder mit der Auflösung $3 \times 227 \times 227$ Pixel verarbeiten und eine Klassifizierung in 1000 Kategorien vornehmen. Da ausschließlich digitale Beschleuniger zum Einsatz kommen, erübrigt sich in diesem Fall eine Analyse der Layer-Robustheit.

	Top. Sort.	HW Eval.	Exploration	Funkt. Eval.
AlexNet	1,10 s	217,12 s	9,62 s	88,45 s
GoogLeNet	1,13 s	1579,05 s	800,40 s	1599,41 s
ResNet-18	0,41 s	794,17 s	61,31 s	286,23 s
ResNet-34	0,91 s	1488,34 s	263,71 s	522,82 s
ResNet-50	1,25 s	1483,10 s	515,51 s	1593,41 s
SqueezeNet V1.1	0,29 s	704,98 s	138,73 s	168,71 s
VGG-11	2,24 s	524,07 s	14,52 s	246,20 s
VGG-13	2,26 s	572,59 s	18,59 s	385,92 s
VGG-16	2,76 s	815,14 s	26,90 s	566,41 s
Yolo v5	0,89 s	935,47 s	491,48 s	589,14 s

Tabelle 5.4: Laufzeitanalyse von CNNParted für verteiltes eingebettetes System

Simulationszeit

Das Ziel von CNNParted ist die zügige Evaluation eines Partitionierungsschemas für die gegebene Systemarchitektur. Folglich ist die Simulationszeit eine wichtige Metrik zur Bewertung der in dieser Arbeit vorgeschlagenen Methodik. Eine detaillierte Laufzeitanalyse der einzelnen Komponenten von CNNParted für die untersuchten Netze ist in Tabelle 5.4 gegeben.

Auffallend an den Ergebnissen ist, dass in fast allen Fällen die Evaluation der Layer für die verschiedenen Beschleuniger des Systems am meisten Zeit beansprucht. Dagegen ist – wenig überraschend – die topologische Sortierung mit durchschnittlich 1,35 s am schnellsten abgeschlossen. Die Laufzeit der Exploration ist insgesamt durch die Komplexität des Netzes sowie der initialen Lösungen bestimmt, weshalb zum Teil sehr große Unterschiede zwischen den CNNs auftreten. Die funktionale Evaluation ist in erheblichem Maße von den Resultaten der Exploration abhängig, da für jedes identifizierte Pareto-optimale Partitionierungsschema die Genauigkeit bestimmt werden muss. Dies resultiert in Unterschieden zwischen den Netzen, die nicht direkt mit der Komplexität korrelieren. Insgesamt ergibt sich mit 3979,99 s (66,33 min) für GoogLeNet die längste Laufzeit. Daraus lässt sich ableiten, dass CNNParted gut geeignet für den Einsatz in der Entwurfsphase eines verteilten eingebetteten Systems ist, da innerhalb einer Stunde selbst etwas größere Netze wie ResNet-50 (59,89 min) hinsichtlich einer geeigneten Inferenz-Partitionierung untersucht werden können.

	Schemata		Part.-Punkte		
	Valide	P.-Optimal	0	1	2
AlexNet	144	22	3	17	2
GoogLeNet	23.306	127	2	92	33
ResNet-18	9.572	47	2	31	14
ResNet-34	32.735	77	2	37	38
ResNet-50	65.193	98	2	51	45
SqueezeNet V1.1	10.049	28	2	20	6
VGG-11	315	42	2	23	17
VGG-13	430	46	2	23	21
VGG-16	512	69	2	31	36
Yolo v5	21.771	78	2	58	18

Tabelle 5.5: Übersicht der Ergebnisse für verteiltes eingebettetes Systems

Ergebnisse & Diskussion

Die Ergebnisse der Untersuchung für ein verteiltes eingebettetes System, bestehend aus drei digitalen Hardwarebeschleunigern, welche vie Ethernet in fester Reihenfolge verbunden sind, zeigt Tabelle 5.5. Sie gibt einerseits die Anzahl der gefundenen validen sowie der Pareto-optimalen Partitionierungsschemata für die untersuchten CNNs an, sowie die Anzahl der vorhandenen Partitionierungspunkte pro Schema.

Die zum Teil geringe Anzahl an validen Lösungen des Optimierungsproblems ist damit zu begründen, dass der Entwurfsraum durch die gegebenen Randbedingungen stark eingeschränkt ist. Folglich ist es für den GA schwierig, gültige Partitionierungsschemata zu finden, weshalb die Ergebnisse an dieser Stelle die Schwächen der verwendeten Heuristik aufzeigen. Abgesehen davon lassen sich dennoch einige interessante Schlüsse aus den Ergebnissen ziehen. So findet CNNParted für die meisten DNNs vorwiegend Pareto-optimalen Schemata mit einem Partitionierungspunkt. Somit führen meist nur zwei Plattformen, der Gemmini- und Eyeriss-Beschleuniger, die Inferenz aus, während die dritte untätig bleibt. Grund hierfür ist insbesondere das verwendete Gigabit Ethernet, welches einen nicht vernachlässigbaren Anteil zur Gesamtlatenz beiträgt. Im Falle zweier Pareto-optimaler Schemata für GoogLeNet beträgt die Linklatenz 3,27 ms bei einer Gesamtlatenz der Inferenz von 36,01 ms und damit 9,08 %. Da die meisten DNNs in der Regel so aufgebaut sind, dass die Anzahl an Elementen der Aktivierungen zu Beginn zunimmt und erst gegen

Ende wieder niedriger ausfällt, sind Aufteilungen in diesem Fall tendenziell zu Beginn und gegen Ende sinnvoll.

Nachfolgende Abbildung 5.14 gibt zusätzliche Einblicke in die Abwägungen, die für ein solches System zwischen den relevanten Metriken (Latenz, Energiebedarf und Durchsatz) erfolgen. Die Gemmini-Architektur ist auf eine geringe Latenz optimiert, während Eyeriss in diesem Fall mehr auf eine geringe Latenz ausgerichtet wurde. Daraus ergibt sich in den meisten Fällen, dass die reiner Ausführung der Inferenz auf dieser Plattformen die geringste Latenz erzielt. Eine ähnliche Beobachtung ergibt sich für Gemmini hinsichtlich des Energiebedarfs. Wie zu erwarten, führt eine partitionierte Inferenz bei sinnvoller Wahl des Partitionierungspunkts zu einem signifikant höheren Datendurchsatz, da die Beschleuniger die Berechnung parallel ausführen. Gleichzeitig verdeutlichen die dargestellten Ergebnisse, dass die Aufteilung der Inferenz auch negative Auswirkungen auf das Gesamtsystem haben kann. Beispielsweise führt die Wahl eines unvorteilhaften Partitionierungspunkts für ResNet-18 zu einer höheren Latenz und höherem Energiebedarf als bei Inferenz auf nur einer Plattform.

5.5.2 Chiplet-basierte Systeme

Im Unterschied zu verteilten eingebetteten Systemen verfügen heterogene Plattformen oder Chiplet-basierte Systeme über ein schnelles und sehr breitbandiges internes Netzwerk, worüber Daten mit geringer Latenz zwischen den Beschleunigern ausgetauscht werden. Aus diesem Grund ist es hier möglich, mehrere, nicht aufeinanderfolgende Partitionen auf eine Plattform zu mappen. Auch entfällt im Vergleich zur vorherigen Untersuchung die Randbedingung, dass die Beschleuniger nur in einer festen Reihenfolge verwendet werden dürfen. Folglich soll diese Anwendungsstudie zeigen, welche Auswirkungen die Erhöhung der Freiheitsgrade auf die Exploration valider und Pareto-optimaler Partitionierungsschemata hat.

Für die nachfolgenden Untersuchungen wird UCIe als Übertragungsprotokoll zwischen den fünf Chiplets angenommen. Zwei davon stellen für die DNN-Inferenz jeweils einen digitalen Beschleuniger bereit, implementiert als Eyeriss- [47] und Simba-ähnliche Architektur [40]. Für beide Plattformen kommt dabei die gleiche Konfiguration wie im Falle der vorherigen Studie zum Einsatz. Die restlichen drei Beschleuniger werden als analoge PIM-Architekturen angenommen, welche sich ebenfalls in ihrer Implementierung unterscheiden. Eine Übersicht der drei Konfigurationen ist in Tabelle 5.6 gegeben. Die nachfolgenden Untersuchungen gehen dabei von MRAM- [188]

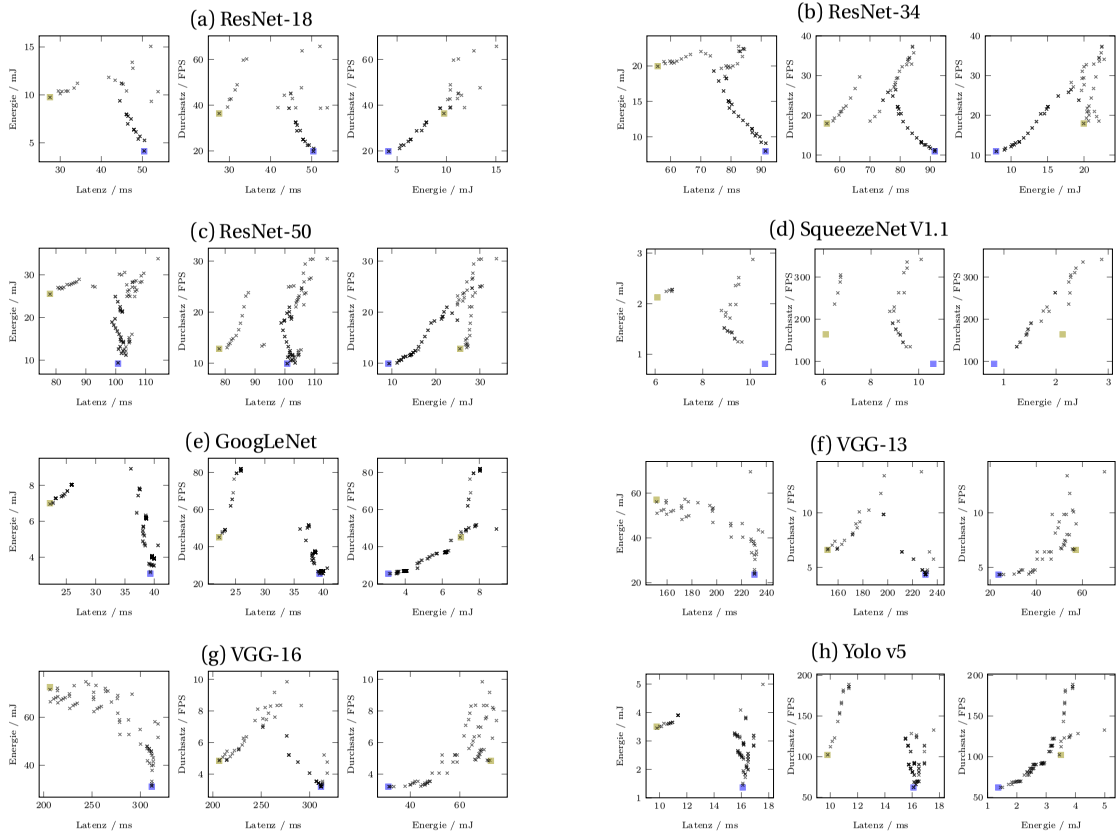


Abbildung 5.14: Pareto-optimale Schemata (×) für verteiltes System verglichen mit Ausführung auf Gemmini (■) bzw. Eyeriss (■)

	PIM 1	PIM 2	PIM 3
Typ	MRAM	MRAM	RRAM
Fläche pro Zelle (μm^2)	0,0456	0,0456	0,0648
Crossbar	256	256	256
DAC Auflösung (bits)		1	
ADC Auflösung (bits)	10	6	8
DAC Num	64	128	256
ADC Num	16	8	16
Inter-Tile Bandbreite (Gbps)	20	20	200

Tabelle 5.6: Für die Evaluation verwendete PIM-Konfigurationen

und RRAM-Architekturen[189] nach dem aktuellen Stand der Technik aus und nehmen eine Fehlerinjektionsrate von 0,01 % an, um SAFs in den analogen Architekturen zu berücksichtigen.

PIM 1 ist ein MRAM-basierter Beschleuniger, der für höchste Genauigkeit optimiert ist. Im Gegensatz dazu bietet die Architektur von PIM 2, die ebenfalls auf der MRAM-Technologie basiert, einen geringen Platzbedarf. Da die ADCs den größten Teil der Fläche einnehmen, verwendet dieser Beschleuniger nur eine ADC-Auflösung von 6 Bit, die jedoch groß genug ist, um eine gute Genauigkeit zu erreichen. PIM 3 ist mit RRAM implementiert und benötigt daher mehr Fläche pro Zelle als die beiden anderen Architekturen. Allerdings verfügt dieser Beschleuniger über eine deutlich höhere Intertile-Bandbreite im internen Network-on-Chip (NoC), weshalb PIM 3 deutlich schneller ist als die beiden anderen PIM-Architekturen. Folglich bilden die drei Plattformen zusammen mit den beiden digitalen Beschleunigern ein breites Spektrum an, um die Partitionierung hinsichtlich unterschiedlicher Metriken zu optimieren.

Laufzeit

Wie bereits in der vorherigen Anwendungsstudie, soll zunächst die Simulationszeit von CNNParted für das gegebenen System näher untersucht werden. Einen Überblick der Laufzeit einzelner Komponenten von CNNParted ist in Tabelle 5.7 gegeben. Da in diesem Fall auch analoge Beschleuniger Teil des eingebetteten Systems sind, erfolgte zusätzlich eine Analyse der Layerrobustheit, deren Laufzeit ebenso aufgeführt ist.

	Top. Sort.	Rob.-Analyse	HW-Eval.	Exploration	Funkt. Eval.
AlexNet	1,11 s	786,30 s	185,25 s	20,67 s	200,33 s
ResNet-18	0,38 s	2773,90 s	598,60 s	72,97 s	1353,56 s
ResNet-34	0,90 s	6550,07 s	1127,75 s	303,32 s	19 110,70 s
ResNet-50	1,25 s	18 483,77 s	1185,85 s	537,56 s	8090,05 s
VGG-11	2,29 s	2273,54 s	439,29 s	33,76 s	4185,36 s
VGG-13	2,31 s	3945,21 s	480,33 s	40,72 s	33 869,41 s
VGG-16	2,84 s	5634,35 s	683,41 s	53,25 s	12 422,80 s

Tabelle 5.7: Laufzeitanalyse von CNNParted für Chiplet-basiertes System

Im Vergleich zur vorherigen Studie wird ersichtlich, dass der zusätzliche Freiheitsgrad bei der Exploration valider Partitionierungsschemata zu einer leicht erhöhten Laufzeit von Exploration und funktionaler Evaluation führt. Demnach ergeben sich mehr valide Lösungen, welche evaluiert werden müssen, bevor der GA eine neue Generation aus der Population erstellt. Anders verhält es sich im Falle der Hardware-Evaluation der zwei digitalen und drei analogen Beschleuniger. Für ResNet-50 benötigt CNNParted hier 1185,85 s, während die Evaluation der Layer für das verteilte eingebettete Systeme und drei digitale Beschleuniger 1483,10 s in Anspruch nimmt. Grund hierfür ist die sequenzielle Ausführung von Timeloop für die Auswertung der digitalen Beschleuniger, weshalb keine parallele Suche nach optimalen Mappings stattfinden kann. Im Gegensatz dazu erlaubt MNSIM eine einfache Parallelisierung der Evaluationen, wodurch die Simulationszeit für die analogen PIM-Beschleuniger keinen direkten Einfluss auf die Gesamtlaufzeit von CNNParted hat.

Des Weiteren ist ersichtlich, dass der zeitliche Aufwand für die Robustheitsanalyse der Layer einen großen Teil der Bearbeitungszeit in Anspruch nimmt. Dies reicht von etwa 13,11 min für ein kleines AlexNet bis 5,13 h für ein ResNet-50. Für letzteres ergibt dies einen Anteil von 65,32 % bei einer Gesamtlaufzeit von 28 297,23 s (ca. 7,86 h). Zudem nimmt die funktionale Evaluation viel Zeit in Anspruch, um die Pareto-optimalen Partitionierungsschemata hinsichtlich der Top-1-Genauigkeit zu untersuchen. In Extremfällen wie dem VGG-13 führt dies zu einer Simulationsdauer von 9,41 h für 551 Pareto-Punkte. Verglichen mit einer Gesamtanzahl von 5,102 validen Schemata wird jedoch offensichtlich, dass der gewählte Ansatz, lediglich Pareto-optimale Punkte hinsichtlich Genauigkeit zu evaluieren notwendig ist, um die Skalierbarkeit des Frameworks zu gewährleisten. Die Ergebnisse belegen somit, dass CNN-

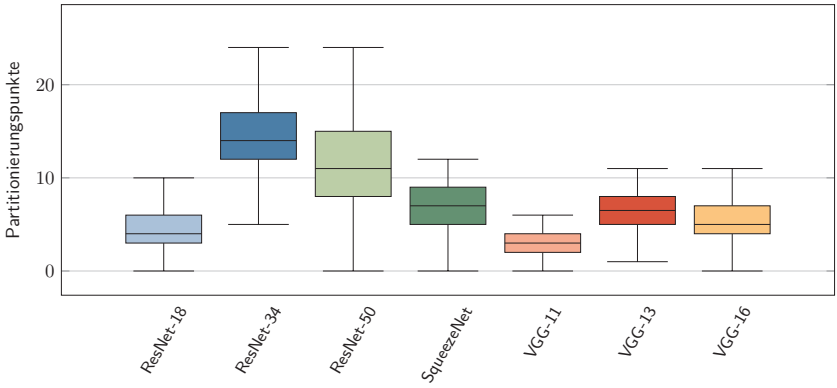


Abbildung 5.15: Verteilung der Anzahl an Partitionierungspunkten pro Pareto-optimalem Schema für Chiplet-basiertes System

Parted gut für den Einsatz in der Entwurfsphase eines eingebetteten Systems basierend auf Chiplets geeignet ist, da innerhalb einiger Stunden selbst größere Netze wie ResNet-50 hinsichtlich einer geeigneten Inferenz-Partitionierung untersucht werden können.

Ergebnisse & Diskussion

Eine Übersicht der Explorationsergebnisse für das gegebenen System mittels CNNParted ist in Abbildung 5.15 dargestellt. In diesem Fall zeigt die Abbildung die statistische Verteilung der Anzahl an Partitionierungspunkten pro Pareto-optimalem Schema für die untersuchten CNNs. Die Ergebnisse weisen dabei einen signifikanten Unterschied zu den Resultaten der Exploration für ein verteiltes eingebettetes System auf. Aufgrund der zusätzlichen Freiheitsgrade und einer höheren Anzahl an Hardwareplattformen findet der GA deutlich mehr valide Lösungen. Darüber hinaus offenbaren die Resultate ein divergentes Bild hinsichtlich der Anzahl an Partitionierungspunkten pro Pareto-optimaler Lösung. Aufgrund der hohen Bandbreite von UCIe gegenüber Gigabit Ethernet, ergeben sich deutlich mehr Partitionierungsschemata mit mehr als einem Partitionierungspunkt. In diesem Anwendungsfall ist gemäß Abbildung 5.15 keine generelle Aussage über eine vorteilhafte Anzahl an Partitionierungspunkten für die Inferenz möglich, da dies stark vom DNN abhängt. Beispielsweise ergab die Suche im Durchschnitt eine höhere Anzahl an

Architektur	Part.-Schema	*acc* (%)	*d* / ms	*e* / mJ	*s* / mm²
ResNet-18	16-bit Digital	77,49	91,78	16,21	2,12
	Latenz-opt.	76,67	43,92	22,29	115,61
	Energie-opt.	77,26	52,04	12,07	20,62
ResNet-34	16-bit Digital	80,52	178,52	33,41	2,12
	Latenz-opt.	80,00	99,93	61,13	129,54
	Energie-opt.	80,37	116,72	24,74	20,622
ResNet-50	16-bit Digital	85,52	251,18	40,81	2,12
	Latenz-opt.	84,62	143,92	43,73	66,48
	Energie-opt.	84,09	146,69	31,09	48,27
SqueezeNet V1.1	16-bit Digital	67,18	13,30	3,17	2,12
	Latenz-opt.	66,64	6,89	7,51	21,18
	Energie-opt.	67,19	8,35	2,30	13,81
VGG-11	16-bit Digital	77,65	255,89	60,99	2,12
	Latenz-opt.	77,62	232,32	43,82	1622,57
	Energie-opt.	77,62	232,32	43,82	1622,57
VGG-13	16-bit Digital	78,44	452,89	85,42	2,12
	Latenz-opt.	78,16	286,01	271,41	1767,19
	Energie-opt.	78,45	356,68	64,32	1644,97
VGG-16	16-bit Digital	79,48	576,75	109,29	2,12
	Latenz-opt.	79,52	268,47	227,41	415,24
	Energie-opt.	79,70	393,76	95,77	308,88

Tabelle 5.8: Ergebnisse für unterschiedliche Optimierungsziele im Vergleich

Partitionen für ResNet-34 im Vergleich zum größeren ResNet-50. Ähnliches ist für VGG-13 im Vergleich zu VGG-16 zu beobachten.

Weitere Erkenntnisse liefert Tabelle 5.8, welche die Resultate der Exploration hinsichtlich einer Optimierung der Latenz und des Energiebedarfs abbildet. Die Ergebnisse belegen, dass die Latenz für die Inferenz erwartungsgemäß enorm von der Verwendung von PIM-Architekturen profitiert. Insbesondere ResNet-18 und VGG-16 profitieren mit einer Latenzreduktion von 52 % bzw. 53,5 % im Vergleich zur reinen Ausführung auf dem digitalen 16-Bit-Eyeriss-Beschleuniger am meisten von der Partitionierung. Es kann festgehalten werden, dass die Latenz auch nach der Optimierung hinsichtlich des Energieverbrauchs in jedem untersuchten Anwendungsfall geringer ausfällt. Darüber

hinaus ist der Platzbedarf im Vergleich zur latenzoptimierten Partitionierung deutlich niedriger, da in diesem Fall aufgrund der hohen Leistungsaufnahme keine Partition auf den größten analogen Beschleuniger (PIM 3) abgebildet wird. In Kombination mit der erreichten DNN-Genauigkeit zeigen diese Ergebnisse die Effektivität der Inferenzpartitionierung in Chiplet-basierten Systemen mittels CNNParted. Zusätzlich liefern die Ergebnisse insbesondere für VGG-11, VGG-13 und VGG-16 den Nachweis, dass die Betrachtung des Flächenbedarfs bei der Partitionierung von DNNs in Systemen bestehend aus PIM-Beschleunigern von hoher Relevanz ist. Der resultierende Platzbedarf für eine latenz- und energieoptimierte Inferenzpartitionierung liegt aufgrund der enorm großen Layer im VGG-13 bei über 1700 mm^2, was deutlich über der realisierbaren Größe eines Chiplets liegt.

Darüber hinaus bieten Abbildungen 5.16 bis 5.18 weitere interessante Erkenntnisse hinsichtlich der Inferenz-Partitionierung. In den Diagrammen werden Pareto-optimale Schemata mit der reinen Ausführung der Inferenz auf dem digitalen 16-Bit-Eyeriss-Beschleuniger verglichen. Die Ergebnisse demonstrieren, dass der Einsatz von PIM-Beschleunigern zu einer deutlichen Verbesserung des EDPs führt, da die Partitionierung mittels CNNParted in allen Fällen eine geringere Latenz bei niedrigerem Energiebedarf erzielt. Dies wird auch am Beispiel des VGG-16 deutlich, welches aufgrund der großen Fully-Connected-Layer hohe Anforderungen an die Rechenleistung des Systems stellt.

Außerdem lässt sich unter anderem anhand der Ergebnisse für ResNet-50 nachweisen, dass auch der Einsatz kleinerer PIM-Beschleuniger zu einer deutlichen Verbesserung des Energieverbrauchs führt. Darüber hinaus demonstrieren die Evaluierungsergebnisse für ResNet-18, dass einige der untersuchten Partitionierungsschemata einen reduzierten Energieverbrauch bei nahezu identischer Top-1-Genauigkeit aufweisen. Darüber hinaus ist festzustellen, dass sich die Latenz von ResNet-18 durch die Auswahl eines geeigneten Partitionierungsschemas bei gleichbleibender Genauigkeit signifikant reduzieren lässt. Dementsprechend führt die hardware-bewusste Partitionierung der Inferenz in solchen heterogenen Multi-Chiplet-Systemen insgesamt zu einer deutlichen Verbesserung der Energieeffizienz bei nur geringem Verlust der Genauigkeit.

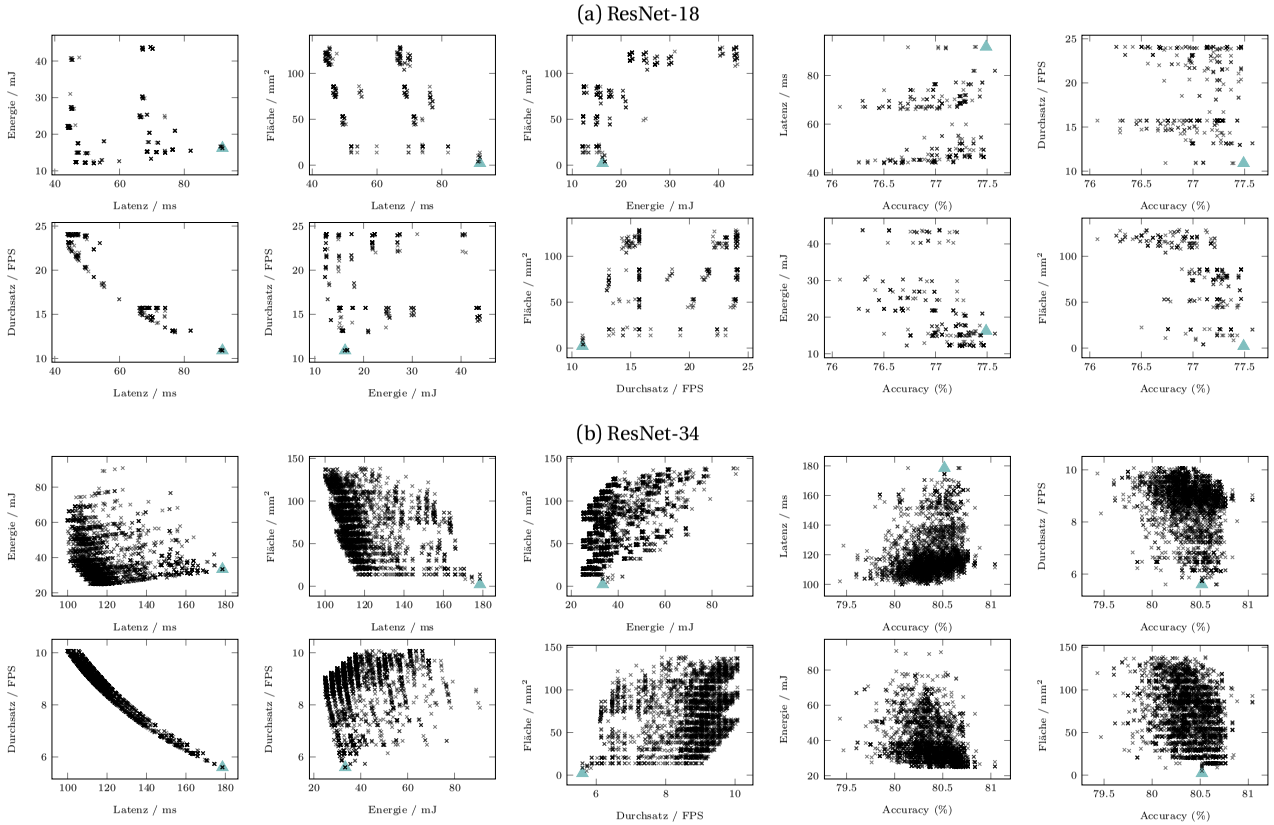


Abbildung 5.16: Pareto-optimale Schemata (×) für ResNet-18 und ResNet-34 im Vergleich zu Referenz (▲)

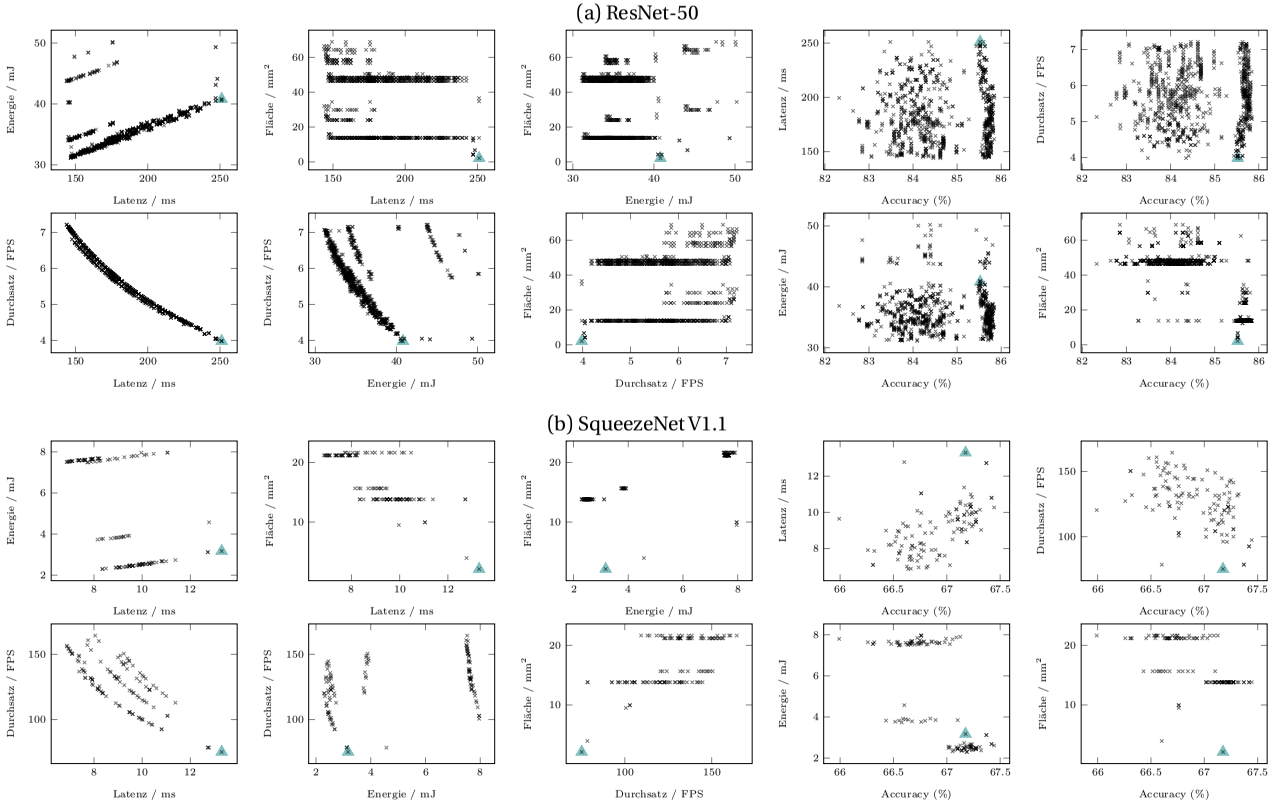


Abbildung 5.17: Pareto-optimale Schemata (×) für ResNet-50 und SqueezeNet V1.1 im Vergleich zu Referenz (▲)

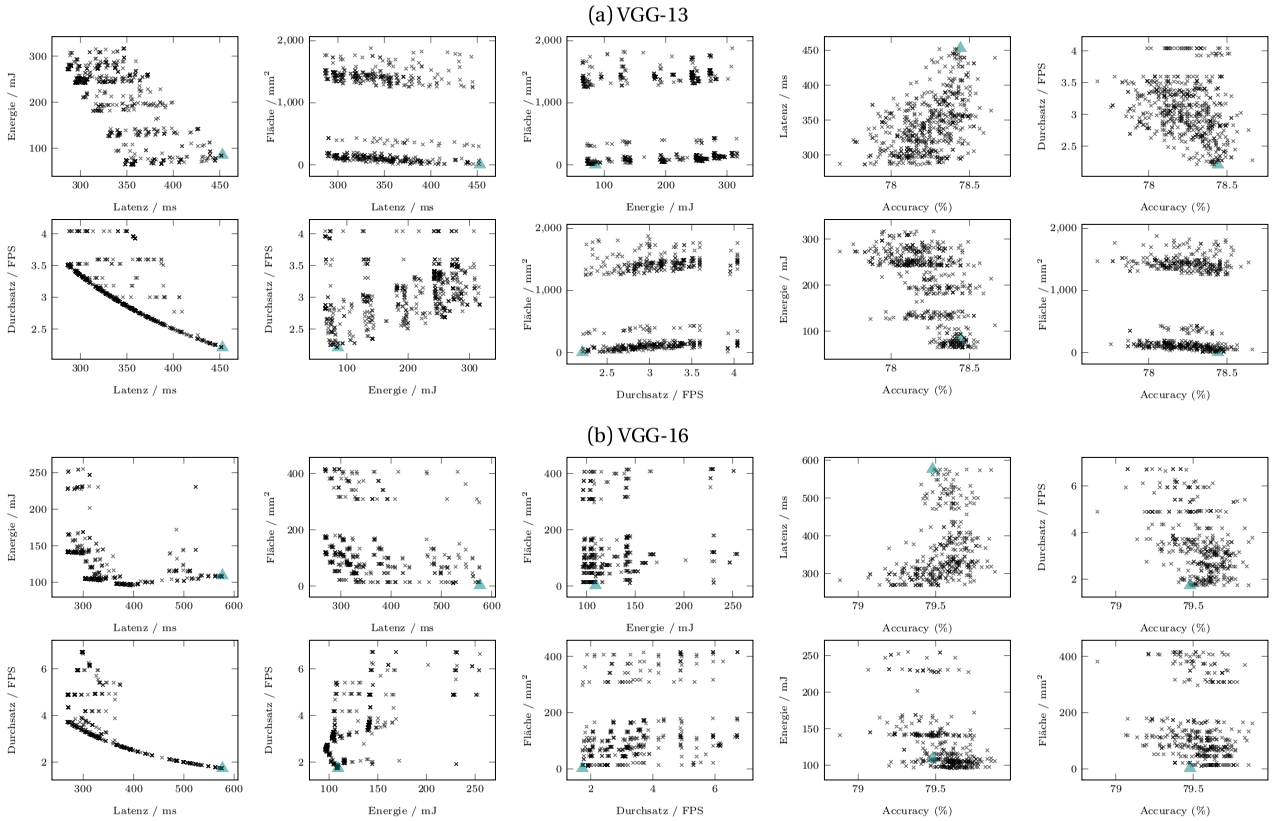


Abbildung 5.18: Pareto-optimale Schemata (×) für VGG-13 und VGG-16 im Vergleich zu Referenz (▲)

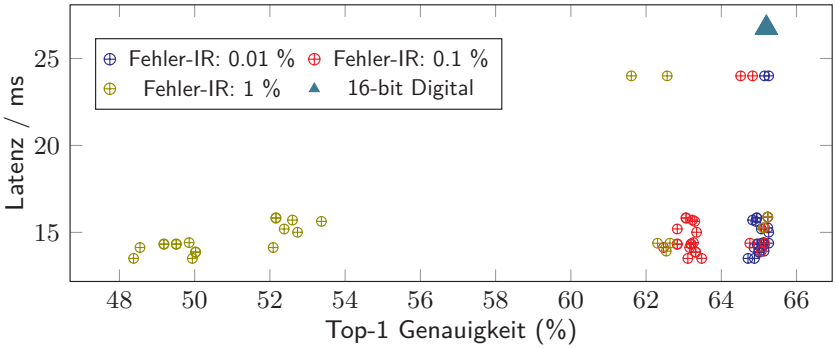


Abbildung 5.19: Genauigkeit von AlexNet für verschiedene Fehlerinjektionsraten (IR)

Variation der Fehlerinjektionsrate

Neben der Bewertung der nicht-funktionalen Metriken ist eine Analyse des Einflusses von PIM-Architekturen auf die Genauigkeit des DNNs notwendig, insbesondere aufgrund der durch SAFs verursachten Abweichungen. In dieser Studie werden nachfolgend drei verschiedene Fehlerinjektionsraten verwendet (0,01 %, 0,1 %, 1 %), um den Einfluss von SAFs genauer zu untersuchen. Die Ergebnisse der Untersuchungen anhand von AlexNet sind in Abbildung 5.19 dargestellt.

Die tiefgreifende Analyse der Ergebnisse liefert zwei interessante Erkenntnisse. Zum einen legen die Resultate nahe, dass die Genauigkeit des Netzes trotz 0,01 % oder 0,1 % vertauschter Bits in den Gewichten nur minimal beeinträchtigt wird. Die Ergebnisse zeigen somit die Effektivität der Layerrobustheitsanalyse und unterstreichen ebenfalls den Vorteil gegenüber modernen Frameworks wie ODiMO [83] oder HybridAC [84]. Diese Frameworks, welche die Exploration eines optimierten Partitionierungsschemas mit einer festen Hardwarekonfiguration durchführen, zeigen signifikante Genauigkeitsverluste für energie- und latenzoptimierte Aufteilungen der Inferenz. Im Gegensatz dazu berücksichtigt CNNParted den Einfluss von SAFs bereits zur Entwurfszeit und kann so eine optimierte Systemarchitektur entsprechend den Anforderungen der Anwendung bestimmen.

Darüber hinaus lässt sich aus den Ergebnissen ableiten, dass PIM-Architekturen mit einer hohen SAF-Rate für den Einsatz in realen Anwendungen eher ungeeignet sind. Dies lässt sich daran verdeutlichen, dass die Genauigkeit bei einer Fehlerinjektionsrate von 0,01 % von 65 %

auf unter 50 % für eine Fehlerinjektionsrate von 1 % sinkt. Demzufolge gibt CNNParted nicht nur eine Rückmeldung über die Systemarchitektur, sondern liefert zusätzlich auch eine Abschätzung der Anforderungen an die Technologie.

Kapitel 6

Hardware/Software Optimierungen

Die Inferenz-Partitionierung von neuronalen Netzen auf verschiedene Hardwareplattformen hat sich, wie bereits zuvor dargelegt, als ein sinnvoller Ansatz erwiesen, um die Performance und die Energieeffizienz des Systems zu erhöhen. In den bisherigen Untersuchungen wurden Anwendungen mit strikten Low-Power-Anforderungen nicht berücksichtigt. In diesem Kapitel erfolgen Untersuchungen solcher Systeme, wobei verschiedene Optimierungsstrategien für die DNN-Inferenz vorgestellt werden.

6.1 Motivation

Das Interesse am Einsatz von IoT hat in den vergangenen Jahren sehr stark zugenommen, sodass dieses heute schon ein elementarer Bestandteil unseres Alltags darstellt. Damit diese Geräte gute Ergebnisse liefern, benötigen sie viele Daten von außen, welche mittels Sensoren erfasst werden. Darüber hinaus werden Algorithmen benötigt, welche die Informationen effizient verarbeiten und die gewünschten Aktionen durchführen. An dieser Stelle findet sich inzwischen verstärkt KI und insbesondere DNNs wieder.

Dabei haben viele Untersuchungen nachgewiesen, dass zeitserienbasierte Anwendungen wie beispielsweise die Handschrifterkennung [190], prädiktive Instandhaltung [191] und EKG-Überwachung [192] besonders von KI profitieren. Die am häufigsten verwendeten DNNs in diesen Bereichen sind LSTMs und TCNs. Allerdings ist die Integration dieser Algorithmen in IoT-Geräte mit mehreren Herausforderungen verbunden, darunter die begrenzten Rechenressourcen, die auf diesen Geräten verfügbar sind. Daher wird die Arbeitslast häufig an leistungsfähigere Systeme oder sogar in die Cloud ausgelagert.

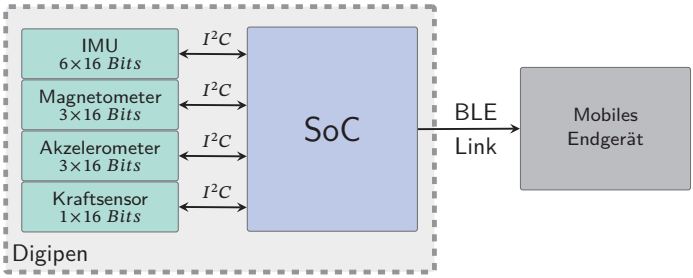


Abbildung 6.1: Beispielhaftes IoT System

Ein solches System ist beispielhaft für die Handschrifterkennung in Abbildung 6.1 dargestellt. Die Rekonstruktion der Handschrift bzw. der Trajektorie einzig anhand der Bewegung des Stifts erfordert die Messung mehrerer physikalischer Größen wie der Beschleunigung und der Rotation. Darüber hinaus muss das System erkennen, ob der Stift die Oberfläche berührt. Infolgedessen verfügt das System über mehrere Sensoren, die im Stift integriert sind. Im Falle des STABILO Digipen gehört dazu eine Inertial Measurement Unit (IMU), ein Magnetometer und ein Kraftsensor nahe der Stiftspitze sowie ein weiterer Beschleunigungssensor auf der gegenüberliegenden Seite [193]. Daraus ergeben sich insgesamt 13 Eingangskanäle, die für die Rekonstruktion der Handschrift bzw. der Trajektorie mit einer Frequenz von 100 Hz bis 400 Hz abgetastet werden.

In solchen IoT-Geräten kommt häufig BLE für die Kommunikation nach außen zum Einsatz. Aus diesem Grund bestehen solche Systeme meist aus einem speziellen BLE-SoC, welcher allerdings stark in Performance und Speichergröße eingeschränkt ist. Ein großer Teil der Ressourcen sind dabei für den Software-Stack des BLE reserviert, wodurch das DNN nur eine geringe Menge an Parametern umfassen darf. Gleichzeitig müssen Latenzanforderungen eingehalten werden, um mit der Verarbeitung von Daten fertig zu sein, bevor die nächsten Messdaten den Prozessor erreichen. Folglich müssen für die Inferenz neuronaler Netze in einem IoT-Gerät nicht nur Optimierungen auf Softwareebene, sondern auch in der Hardware erfolgen, um den Anforderungen der Anwendung gerecht zu werden.

Bei Betrachtung des Gesamtsystems stellt allerdings der Link zwischen dem IoT-Gerät und dem mobilen Endgerät bzw. der Cloud die kritischste Komponente dar. BLE eignet sich sehr gut für den Einsatz in solchen Anwendungsszenarien, da es ein Low-Power-Kommunikationsprotokoll ist und viele Endgeräte dieses unterstützen. Jedoch ist der maximal mögliche Datendurchsatz,

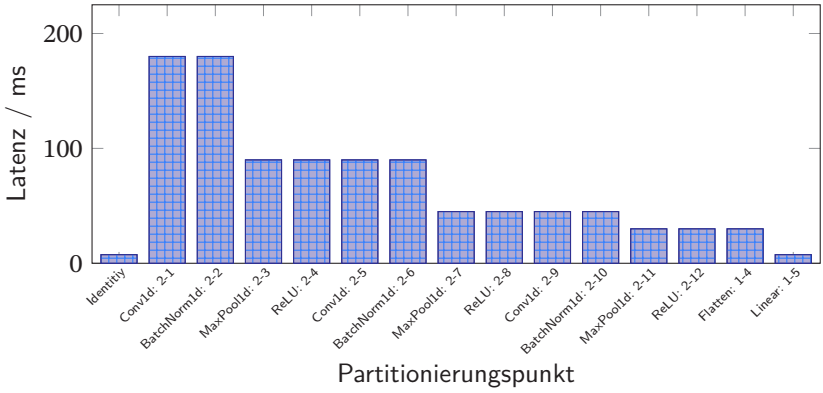


Abbildung 6.2: BLE Link-Latenz für die Partitionierung eines TCN

wie von Tosi et al. [194] demonstriert, mit 236,7 kbit/s im fehlerfreien Fall stark beschränkt. Dies wirkt sich unmittelbar auf die Partitionierung eines DNN in solchen Systemen aus.

Im Folgenden soll ein TCN für die Handschrifterkennung beispielhaft hinsichtlich der Partitionierung untersucht werden. Dabei handelt es sich um ein DNN bestehend aus 14 Layern, wie von Wehbi et al. vorgeschlagen [195]. Schon die Betrachtung der Latenz für eine Übertragung der Zwischenergebnisse mittels BLE, wie in Abbildung 6.2 dargestellt, legt nahe, dass eine Aufteilung der TCN-Inferenz auf zwei Plattformen nicht sinnvoll funktioniert. Insbesondere die Größe der Output Feature Maps der ersten Layer des Netzes führt dazu, dass allein durch die Übertragung dieser zum mobilen Endgerät eine hohe Latenz zustande kommt. Lediglich die Übertragung der Rohdaten bzw. des Ergebnisses sowie der Aktivierungen der hinteren Layer liegt mit 7,5 ms bzw. 30 ms in einem realisierbaren Bereich.

Dieses Anwendungsbeispiel veranschaulicht, dass die Inferenz-Partitionierung nicht in jedem Fall sinnvoll ist. Insbesondere kleine Netze, wie diese häufig in IoT-Anwendungen zum Einsatz kommen, sind in Kombination mit BLE nicht für eine Aufteilung der Layer über mehrere Geräte geeignet. Dabei ist die Verarbeitung der Daten im IoT-Gerät vorteilhaft, da hierdurch eine Unabhängigkeit vom Endgerät erreicht wird. Eine unterbrechungsfreie Verbindung muss folglich nicht bestehen, um einen Datenverlust zu verhindern. Für die energieeffiziente und performante Inferenz unter den gegebenen Anforderungen der Anwendung sind daher Optimierungen der DNN-Architektur und der Hardware erforderlich.

6.2 DNN Optimierungen

Die stark limitierte Bandbreite sowie der geringe Speicher stellen das größte Hindernis bei der Integration von DNNs in IoT-Geräte dar. Um die Parameter dieser Netze zu komprimieren, wird in der Regel auf Quantisierung zurückgegriffen [196], welche nebenbei auch zu einer Verringerung der Rechenkomplexität beiträgt. Für Anwendungen wie die Trajektorienrekonstruktion anhand von IMU-Daten ist die benötigte Bandbreite gering, da hierbei lediglich eine Übertragung des rekonstruierten Vektors erfolgt. Im Falle der Handschrift- oder Spracherkennung wird dagegen eine Klassifizierung vorgenommen. Dabei gibt es effiziente Ansätze wie die Connectionist Temporal Classification (CTC) [197], bei der die Auftrittswahrscheinlichkeit der möglichen Buchstaben und Zeichen vom DNN als Ergebnis ausgegeben werden. Die nachfolgenden Erläuterungen und Ergebnisse der Optimierung einer solchen Anwendung wurden vom Autor in [Kre22b] veröffentlicht.

Beispielhaft kommt ein LSTM für die Handschrifterkennung mit CTC von Wehbi et al. [198] zum Einsatz, wobei lediglich die erforderliche Dekodierung zur Worterkennung auf dem mobilen Endgerät stattfindet. Die Ausgabe des Modells, auch als CTC-Matrix bezeichnet, besteht aus 53 Label-Auftrittswahrscheinlichkeiten, welche 26 Groß- und Kleinbuchstaben sowie ein spezielles CTC-Symbol ('-') darstellen. Eine oder mehrere solcher Matrizen entsprechen einem Zeichen. Das CTC-Symbol wird verwendet, um zwischen den Buchstaben eines Wortes zu unterscheiden. So ergibt sich aus der Ausgabe 'aaa' → 'a' bzw. 'aa-a' → 'aa'. Für die finale Auswertung muss schließlich eine CTC-Dekodierung mittels eines entsprechenden Algorithmus erfolgen [199]. Als Metrik für die Genauigkeit des Netzes kommt die Character Error Rate (CER) zum Einsatz. Diese beschreibt den Anteil falsch klassifizierter Zeichen in Verhältnis zur Gesamtzahl an erkannten Zeichen. Die CER basiert auf der Levensthein-Distanz, welche den Unterschied zwischen zwei Wortsequenzen misst [200]. Dementsprechend ergibt sich die CER aus der Summe der Levensthein-Distanzen geteilt durch die Anzahl aller Zeichen im Testdatensatz. Nachfolgend werden zwei Ansätze für die CTC-Dekodierung verwendet: *Best Path* sowie *Beam Search* mit und ohne Language Model (LM).

Erstgenannter Algorithmus nutzt die Klasse mit der höchsten Auftrittswahrscheinlichkeit pro Zeitschritt für die Worterkennung, wodurch nicht die gesamte CTC-Matrix, sondern nur das beste Ergebnis an das mobile Endgerät zur finalen Verarbeitung übertragen werden muss. Bei der Beach Search werden verschiedene mögliche CTC-Pfade (Beams) evaluiert, um herauszufin-

den, welcher Buchstabe am wahrscheinlichsten geschrieben wurde. Dabei wird die Zahl der parallel verfolgten Pfade anhand eines Parameters (Beam Width) frei konfiguriert. Die Suche nach dem korrekten Wort kann darüber hinaus durch die Verwendung eines LMs optimiert werden [201]. Hierbei wird die Auftrittswahrscheinlichkeit eines bestimmten Buchstabens in einer Zeichensequenz, und somit die Sprachcharakteristika, in die Zeichenerkennung einbezogen. Der Einfluss des LMs auf die Entscheidung im Dekoder und damit auf die Genauigkeit der Handschrifterkennung wird anhand eines Gewichtungsfaktors gesteuert.

6.2.1 Bandbreiten-Optimierung

Werden alle Sensoren mit einer Frequenz von 100 Hz abgetastet, ergibt sich für die Übertragung der 16-Bit Rohdaten über BLE eine benötigte Bandbreite von 20,31 kbit/s. Trotz der höheren Anzahl an Elementen in der CTC-Matrix verglichen zu der Anzahl der Sensorkanäle, liegt die Datenrate für die Ausgabe des DNNs mit 20,70 kbit/s nur minimal darüber. Grund hierfür sind die auf die zeitliche Dimension angewandten Pooling-Layer, wodurch Ergebnisse mit einer Frequenz von 12,5 Hz und damit nur alle 80 ms generiert werden. Folglich ist die Ausführung der Inferenz auf dem Stift ohne weitere Optimierungen nicht sinnvoll. Durch geeignete Wahl des CTC-Dekodierungsalgorithmus kann die Zahl der Klassen und damit der Elemente reduziert werden. Im Falle von *Best Path*, welcher auch ursprünglich von Wehbi et al. [198] angewandt wurde, erfolgt die Übertragung von nur einer Klasse. Damit lässt sich eine CER von 15,7 % bei einer benötigten Bandbreite von 0,39 kbit/s erzielen.

Mittels *Beam Search* kann eine höhere Klassifizierungsgenauigkeit erreicht werden [199], allerdings auf Kosten einer höheren Bandbreite. Abbildung 6.3 zeigt die Evaluationsergebnisse der Genauigkeit für *Best Path* und *Beam Search* mit verschiedenen LM-Faktoren in Abhängigkeit der Beam Width (konstant für *Best Path*). Für *Beam Search* werden Bean Widths von 1 bis 30 dargestellt, da größere Werte zu keiner signifikanten Verbesserung der CER beitragen. Im Falle der Verwendung von *Beam Search* ohne LM ergibt sich unabhängig der Beam Width keine Verbesserung im Vergleich zu *Best Path*. Dagegen ist festzustellen, dass der Einsatz eines LM ab einer Beam Width von vier zu einer deutlichen Reduzierung der CER führt. Mit 11,19 % erzielt das DNN mit *Beam Search* bei einer Beam Width von 22 und einem LM-Faktor von zwei die höchste Genauigkeit. Den besten Kompromiss zwischen benötigter Bandbreite und CER stellt die Konfiguration mit elf Beams

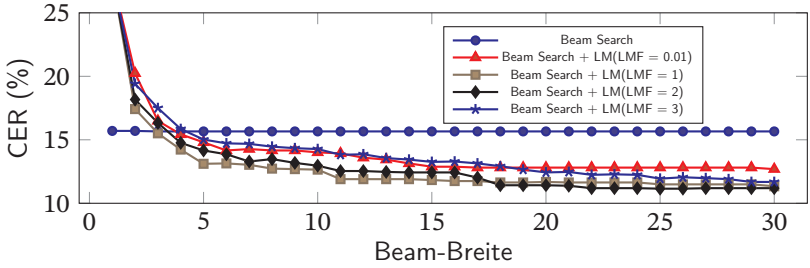


Abbildung 6.3: Auswirkung verschiedener Beam Widths auf die CER

und einem LM-Faktor von eins dar (CER = 11,90 %). Daraus resultiert eine Datenübertragungsrate von 4,30 kbit/s und damit eine signifikante Reduzierung der benötigten Bandbreite um 78,83 %. Folglich kann die Ausführung der DNN Inferenz im Stift in Kombination mit CTC-Dekodierung im mobilen Endgerät erfolgen und erzielt dabei auch eine Energieeinsparung bei der Datenübertragung.

6.2.2 Modell-Quantisierung

Um die Inferenz im Stift trotz des geringen Speichers grundsätzlich möglich zu machen und hinsichtlich Latenz und Energiebedarf zu optimieren, ist in einem weiteren Schritt eine Quantisierung der Gewichte des DNNs erforderlich. Für die nachfolgende Studie wurden 8-Bit Gewichte angenommen, welche das Ergebnis der PTQ sind. Die Aktivierungsfunktionen sowie die Aktivierungen betraf diese Quantisierung nicht, sie basieren auf 32-Bit Fließkommazahlen.

Grund dafür ist, dass die Gewichte den Großteil des Speicherbedarfs der Anwendung ausmachen. Nicht quantisiert benötigen diese bereits 2,80 MB in einem nichtflüchtigen Speicher. Daneben müssen zusätzlich Zwischenergebnisse, die Zustände der LSTM-Zellen sowie der Programmcode gespeichert werden. Diese machen allerdings einen geringeren Anteil aus, wie in Abschnitt 6.2.3 gezeigt.

Abbildung 6.4 stellt den Vergleich zwischen quantisiertem sowie nichtquantisiertem Netz mit und ohne LM in Abhängigkeit der Beam Width grafisch dar. In beiden Fällen hat die Quantisierung der Gewichte keinen nennenswerten Einfluss auf die CER und damit die Genauigkeit des DNNs. Dagegen reduziert die Quantisierung der Gewichte die Größe des Modells

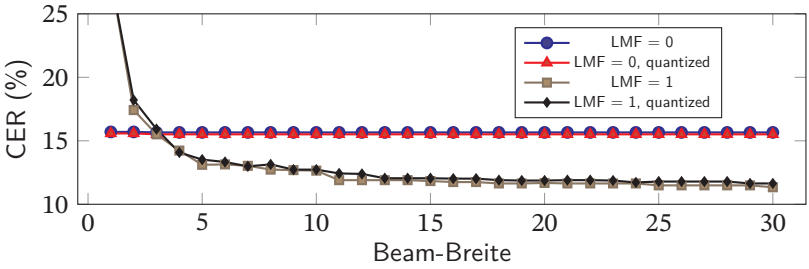


Abbildung 6.4: Auswirkung verschiedener Beam Widths und Gewichtsquantisierung auf die CER

von 2,80 MB auf 0,74 , folglich um 73,57 %. Dies ermöglicht den Einsatz des Netzes auf mehr Hardwareplattformen und sorgt gleichzeitig für eine Reduzierung des Energiebedarfs durch die geringere Anzahl auszulesender Bits im Speicher.

6.2.3 Hardware-Anforderungen

Der Speicher benötigt viel Platz im SoC und ist folglich eine kritische Komponente im Hinblick auf die Kosten, insbesondere für IoT-Anwendungen. Geht man von einer sequenziellen Abarbeitung des DNNs aus, definieren die größte Input und Output Feature Map den maximalen dynamischen Speicherbedarf. Im Falle des DNNs für die Handschrifterkennung verarbeitet das zweite 1D-CONV-Layer einen $N/2 \times 512$ großen Tensor und gibt $N/2 \times 256$ Werte aus. Dabei entspricht die minimale Anzahl an Zeitschritten N zur Generierung einer validen CTC-Matrix 24. Diese Zahl resultiert aus den drei Pooling-Layern (Fenstergröße zwei) und den drei 1D-CONV-Layern mit den jeweiligen Kernelgrößen fünf, drei und drei. Um diese Daten als 32-Bit Fließkommazahlen zwischenzuspeichern, werden folglich 36,86 kB benötigt. Hinzu kommen 131,07 kB an Speicherplatz für die beiden bidirektionalen LSTM-Layer mit jeweils 64 Zellen und 128 Eingängen.

Zur Bestimmung der Speicheranforderung fehlt abschließend noch die Größe des Programmcodes, welche wesentlich von der verwendeten Instruction Set Architecture (ISA) abhängt. Daher werden im Folgenden verschiedene Architekturen hinsichtlich der Programmgröße für das gegebene DNN untersucht. Als Grundlage der Analyse dient das quantisierte DNN und die RISC-V

	ARMv7E-M	RV32IC	RV32IMC	RV32IMFC
Programmgröße	105,28 KB	305,58 KB	302,86 KB	291,90 KB
Anzahl Instruktionen	$0,73 \cdot 10^9$	$6,75 \cdot 10^9$	$1,29 \cdot 10^9$	$0,34 \cdot 10^9$
Zielvorgabe	> 9,13 GFLOPS	> 84,38 GOPS	> 16,13 GOPS	> 4,25 GFLOPS

Tabelle 6.1: Programmgrößen und Performanceanforderungen für ARMv7E-M und drei RISC-V ISAs

ISA aufgrund ihrer Erweiterbarkeit, welche eine einfache Abbildung unterschiedlicher Hardwarearchitekturen ermöglicht. Bei der Übersetzung des Programmcodes kommt ein RISC-V Crosscompiler zum Einsatz, welcher einfache Integer-Operationen (I), Integer Multiplikationen und Division (M) und komprimierte Instruktionen (C) sowie einfache Fließkomma-Operationen (F) und weitere unterstützt. Daraus lassen sich verschiedene Implementierungsvarianten ableiten, nachfolgend werden RV32IC, RV32IMC und RV32IMFC für die Abschätzung der Programmgröße herangezogen. In den ersten beiden ISAs werden Fließkomma-Instruktionen als Soft-Float-Operationen implementiert, wodurch insgesamt mehr Befehle für die Ausführung der Inferenz benötigt werden. Zusätzlich wird nachfolgend die ARMv7E-M ISA betrachtet, welche häufig in IoT-SoCs zum Einsatz kommt. Tabelle 6.1 zeigt die Ergebnisse der Untersuchungen. Da der RISC-V Compiler derart konfiguriert wurde, dass er das Programm auf die Speichergröße anstatt der Latenz optimiert, ergeben sich für die drei RISC-V ISAs nur geringe Unterschiede. Dagegen hat die ARMv7E-M ISA deutlich weniger Speicherplatz, aufgrund der darin enthaltenen zusätzlichen Instruktionssatz-Erweiterungen. Summiert man schließlich die unterschiedlichen Speicheranforderungen auf, ergibt sich ein minimaler Verbrauch von 1,2 MB, welcher für die Ausführung des DNNs in Hardware bei Verwendung der RV32IMFC ISA benötigt wird.

Neben dem Speicherplatz muss das System außerdem über eine ausreichend hohe Performance verfügen, um die DNN Inferenz rechtzeitig vor dem Eintreffen neuer Daten abzuschließen. Nach dem Einlesen der ersten 24 Zeitschritte, wird diese im Falle des DNNs nach jedem achten Zeitschritt neu ausgeführt. Zusammen mit der Abtastfrequenz von 100 Hz ergibt sich daraus eine maximale Latenz von 80 ms, um eine Verarbeitung der Sensordaten ohne Verlust von Informationen zu gewährleisten. Um die Anforderungen an die Hardware zu analysieren, wurden die Anzahl der auszuführenden Instruktionen für jede ISA mittels OVPsim [202] für ARMv7E-M bzw. Spike [203] für die RISC-V Architekturen ermittelt. Aus diesen Ergebnissen lassen sich, wie in Tabelle 6.1 dargestellt, die minimalen Hardwareanforderungen für jede ein-

zelne ISA hinsichtlich Performance ableiten. Dabei ergibt sich aufgrund der hohen Anzahl an Instruktionen für die RV32IC-ISA, dass diese für den Einsatz in IoT-Anwendungen ungeeignet ist. Gleiches gilt für RV32IMC. Obwohl diese im Vergleich deutlich weniger Instruktionen benötigt, erfüllt derzeit kein auf dem Markt verfügbarer SoC die Anforderungen von 16,13 GOPS für eine maximale Latenz von 80 ms. Ähnlich dazu ist gegenwärtig kein ARM Cortex-M4F-basiertes System zu finden, welches genügend On-Chip Speicher für die DNN-Inferenz bietet. Dagegen existieren, ausgehend vom aktuellen Stand der Technik, auf der RV32IMFC-ISA basierende Systeme, welche die Performance- und Speicheranforderungen, wie in Tabelle 6.1 dargestellt, einhalten [204]. Demnach ist eine Ausführung des quantisierten DNNs zur Handschrifterkennung in IoT-Geräten bei geeigneter Wahl der ISA grundsätzlich möglich.

6.3 Approximate Computing

Problematisch bei der Verwendung von Mikrocontrollern, welche nicht für eine bestimmte Art von Aufgabe optimiert wurden, ist die geringere Energieeffizienz bzw. Performance sowie ein erhöhter Platzbedarf des Systems. Um diese Probleme zu umgehen, kommen häufig spezialisierte Hardwarebeschleuniger zum Einsatz, welche den Datenfluss ressourcenschonender auf die Architektur abbilden und dabei gleichzeitig Speicherzugriffe reduzieren. In diesem Abschnitt wird der LSTM-Beschleuniger ATLAS vorgestellt, welcher approximative Methoden nutzt, um insbesondere die Matrix-Multiplikationen sowie die Aktivierungsfunktionen ressourceneffizient in Hardware umzusetzen. Die Architektur zielt auf einen Einsatz als Co-Prozessor im eingebetteten System ab, wodurch die Inferenz des Netzes unabhängig von der Datenübertragung mittels BLE erfolgt. Die nachfolgenden Erläuterungen und Evaluationen für ATLAS wurden vom Autor in [Kre23c] veröffentlicht.

6.3.1 Hintergrund

Im Gegensatz zu anderen DNN-Architekturen wie beispielsweise CNNs, stellt die Implementierung der Inferenz von LSTMs in Hardware eine deutlich größere Herausforderung dar. Gemäß Rybalkin et al. [205] gibt es dafür drei wesentliche Gründe. Zum einen wirken sich zeitliche Rekursionen und die Abhängigkeit der Daten von früheren Zeitschritten negativ auf den Gesamtdurchsatz eines Beschleunigers aus. Zweitens lassen sich die nicht-linearen

Aktivierungsfunktionen nicht effizient in Hardware umsetzen, sondern müssen approximiert werden, um eine akzeptable Hardwareressourcennutzung und eine hohe Energieeffizienz zu erzielen. Schließlich ergibt sich eine deutlich höhere Anforderung an den Speicherplatz aufgrund der zu sichernden Zellzustände des LSTMs. Folglich ist die maximale Performance in der Regel durch die verfügbare Speicherbandbreite limitiert.

Es gibt verschiedene Möglichkeiten, diese Herausforderungen anzugehen. Wie zuvor erwähnt und auch von verschiedenen Forschungsgruppen nachgewiesen [110, 206], ermöglicht Quantisierung von Gewichten und Aktivierungen die Komprimierung der Modelle. Dabei nutzen LSTM-Netze in eingebetteten Systemen meist 8–16-Bit Fixed-Point-Repräsentation für die Inferenz. Diese führt nicht nur zu einer Reduzierung der benötigten Speicherbandbreite, sondern verringert auch die Rechenkomplexität.

Allgemein betrachtet besteht die Inferenz aus drei wesentlichen mathematischen Operationen: MVM, elementweises Vektorprodukt und elementweise nicht-lineare Aktivierungsfunktionen. In den letzten Jahren wurden mehrere Hardware-Architekturen vorgeschlagen, die auf eine Optimierung der Energieeffizienz und der Rechenleistung von LSTM-Beschleunigern abzielen. Mit Fokus auf den rechenintensivsten Teil, die MVMs, schlagen Conti et al. [207] den Hardwarebeschleuniger Chipmunk vor, welcher auf einem systolischen Array bestehend aus MAC-Einheiten basiert. Diese Architektur ermöglicht die parallele Berechnung mehrerer Multiplikationen, woraus ein deutlicher Leistungszuwachs resultiert. Die Aktivierungsfunktionen wurden als LUTs implementiert. Als problematisch erweist sich hierbei die systolische Architektur des Beschleunigers, welche einen hohen Flächen- und Energiebedarf zur Folge hat.

Azari et al. [208] schlagen ein wesentlich effizienteren LSTM-Beschleuniger namens ELSA vor, welcher approximative Multiplizierer für die Berechnung der MVMs nutzt. Diese basieren auf einer Finite State Machine (FSM) und sorgen für eine verringerte Leistungsaufnahme bei nur geringem Einfluss auf die Genauigkeit des DNNs. Allerdings geht diese Architektur auf Kosten der Latenz und damit auch der Performance, da die FSM mehrere Zyklen für die Berechnung der Multiplikation benötigt. Darüber hinaus wird die Anzahl der notwendigen Takte adaptiv zur Laufzeit angepasst, was einen speziellen Controller für die Synchronisation mit den anderen Modulen des Beschleunigers erfordert.

Neben verschiedenen ASIC-Realisierungen, existieren auch einige Untersuchungen, welche die Implementierung eines solchen Beschleunigers auf einem FPGA untersucht haben [209, 210, 211]. Allerdings zielen diese meist

auf eine hohe Leistungsfähigkeit ab, woraus ein unangemessen hoher Energiebedarf für batteriebetriebene Geräte resultiert. So schlagen beispielsweise Bank-Tavakoli et al. den LSTM-Beschleuniger POLAR vor [212], welcher zur Entwurfszeit hinsichtlich der Parallelisierung von MVM-Berechnungen konfiguriert und somit optimal an die Anwendung und die Plattformbeschränkungen angepasst ist.

Einen größeren Fokus auf die Energieeffizienz legen Chen et al. mit ihrem Low-Power LSTM-Beschleuniger Eciton [213], optimiert für einen IoT-kompatiblen FPGA. Durch die Verwendung einer 8-Bit Quantisierung sowie der Implementierung von stückweise linearen Funktionen zur Realisierung der Aktivierungsfunktionen erzielt die Architektur eine geringere Rechenkomplexität und ermöglicht somit eine einfachere logische Umsetzung auf den FPGA. Allerdings führt dieser Ansatz zu einer deutlich reduzierten Genauigkeit des DNNs und ist folglich trotz der hohen Energieeffizienz nur eingeschränkt in IoT-Anwendungen einsetzbar.

Zuletzt haben Qian et al. eine energieeffiziente LSTM-Hardwarearchitektur für einen kleinen FPGA vorgeschlagen [214]. Der Beschleuniger verwendet fünf parallele Einheiten für die Ausführung von Multiplikations- und Akkumulationseinheiten und separate LUTs für jede Art von Aktivierungsfunktion. Dies ermöglicht die parallele Berechnung der Werte für jedes Gatter und erhöht somit die Leistungsfähigkeit des Systems. Jedoch kommen LUTs mit einer Tiefe von 256 zum Einsatz, was zu einem hohen Ressourcenverbrauch führt. Zusätzlich geht die Architektur vom Einsatz von acht DSPs im FPGA aus, welche nicht in jeder Plattform zur Verfügung stehen. Gleichzeitig sorgen die DSPs zwar für eine hohe Rechengenauigkeit, allerdings erzielen diese keine hohe Energieeffizienz für Rechenoperationen mit quantisierten Daten, wie von Boutros et al. nachgewiesen [215]. Somit fehlt es an einer geeigneten Lösung für die energieeffiziente und performante LSTM-Inferenz in IoT-Geräten.

6.3.2 Hardwarekonzept

Zeitserienbasierte IoT-Anwendungen wie die Handschrifterkennung sind in der Regel stark in der maximalen Leistungsaufnahme und der Fläche limitiert, müssen gleichzeitig aber eine gewisse Performance für den Einsatz von KI bieten. Der im Folgenden vorgeschlagene LSTM-Beschleuniger ATLAS zielt darauf ab, diese Anforderungen einzuhalten. Eine Übersicht der Architektur ist in Abbildung 6.5 dargestellt.

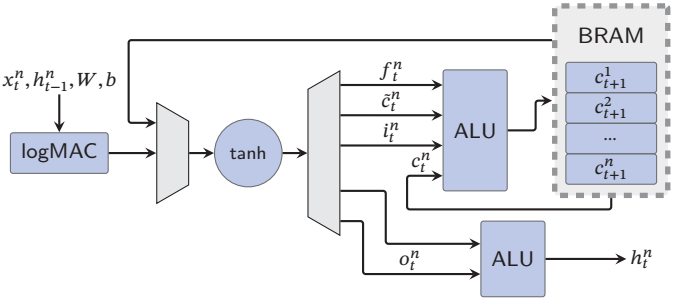


Abbildung 6.5: Übersicht der vorgeschlagenen Architektur des LSTM-Beschleunigers

Das Design von ATLAS leitet sich direkt aus der Struktur einer LSTM-Zelle ab, wodurch die Hardware folglich sehr gut an den Datenfluss angepasst ist. Zunächst beinhaltet der Beschleuniger eine approximative MAC-Einheit zur Berechnung der MVMs. Da dieser den binären Logarithmus für die Implementierung der Multiplikation ausnutzt, werden für diese Operation lediglich Additionen sowie bitweise Schiebeoperationen durchgeführt. Folglich entfällt auch der Bedarf an DSPs oder ähnlichen Komponenten zur effizienten Realisierung der MAC-Operationen. Um einen geringen Ressourcenverbrauch zu erzielen, nutzt der Beschleuniger eine einzige Komponente für die Implementierung der Aktivierungsfunktionen, wobei der Datenfluss über einen Demultiplexer und einen Multiplexer gesteuert wird. Ein interner Controller, welcher als FSM realisiert ist, konfiguriert diese entsprechend der von außen gegebenen LSTM-Layer-Parameter. Außerdem sorgen zwei Arithmetisch-Logische Einheiten (ALUs) für die Implementierung der einfachen Additionen in Hardware und ein BRAM-Modul für die Speicherung der Zellzustände.

Die gesamte Architektur ist flexibel gestaltet und unterstützt verschiedene Bitbreiten der Eingangsdaten sowie Gewichte. Dies ermöglicht die Auswahl eines geeigneten Quantisierungsschemas auf der Grundlage der Systemanforderungen und Hardwarebeschränkungen. Neben den konfigurierbaren Bitbreiten umfasst dies auch die Auswahl der Approximation der Aktivierungsfunktion, die je nach Anwendung erfolgt.

Datenfluss-Controller

Um die Wiederverwendung der Aktivierungsfunktionseinheit für alle LSTM-Gates zu ermöglichen, muss der Datenfluss durch den Beschleuniger geleitet

werden. Folglich ist zur Steuerung des Multiplexers und des Demultiplexers eine Komponente erforderlich, die die entsprechenden Signale für diese Module erzeugt. Nach dem Start der Inferenz führt ATLAS die Multiplikation der Eingänge mit den Gewichten gemäß der LSTM-Charakteristika durch. Außerdem wird der Demultiplexer so konfiguriert, dass er die entsprechende Aktivierungsfunktion auf das Ergebnis der Multiplikation anwendet. Gleichzeitig konfiguriert der Controller den sich anschließenden Multiplexer derart, dass er den Ausgang der Aktivierungsfunktion an eine der beiden ALUs zur weiteren Verarbeitung weiterleitet.

Innerhalb des Controllers ist ein Zähler implementiert, der das Ende der laufenden Berechnungen erkennt und entsprechend in den nächsten Zustand wechselt. Sobald alle Multiplikationen eines LSTM-Gates ausgeführt worden sind, passt der Datenfluss-Controller die Konfiguration des Demultiplexers an, um das nächste LSTM-Gate zu verarbeiten. Sobald die Ergebnisse aller LSTM-Gates vorliegen, wird der Zellzustand aktualisiert und schließlich die Ausgabe der LSTM-Zellen erzeugt.

Approximativer Multiplizierer

Die Verarbeitung einer ganzen LSTM-Schicht umfasst viele Multiplikationen, die den größten Teil der Zeit während einer Inferenz beanspruchen. In der Architektur von ATLAS werden solche Berechnungen durch eine Reihe von Additionen, Subtraktionen und Verschiebeoperationen durchgeführt. Um die Anzahl der für eine einzige Multiplikation erforderlichen Operationen drastisch zu reduzieren, verwendet ATLAS dabei den binären Logarithmus wie von Mitchell vorgeschlagen [216]. Anstatt beide Werte zu multiplizieren, wird das Ergebnis so durch eine Addition ermittelt.

$$a \cdot b = 2^{log_2(a \cdot b)} = 2^{log_2(a)+log_2(b)} \tag{6.1}$$

Mitchells Algorithmus nutzt eine stückweise Polynomfunktion zur Approximation des Logarithmus. Dies basiert auf folgender Umschreibung des binären Logarithmus

$$log_2(N) = k + log_2(1 + x), \tag{6.2}$$

wobei k der Stelle der führenden Eins und x der Mantisse entspricht. Da $x << 1$ gilt, wird folgende Approximation vorgenommen:

$$log_2(1 + x) \approx x \tag{6.3}$$

Auf Grundlage dieser Annahmen wird die Multiplikation zweier Zahlen N_1 und N_2 demnach wie folgt angenähert:

Definition 6 (Approximative Multiplikation)**.** *Die approximative Multiplikation zweier reeller Zahlen N_1 und N_2 basierend auf Mitchells Algorithmus ist gegeben durch*

$$log_2(N_1 \cdot N_2) \approx k_1 + k_2 + x_1 + x_2$$

Für die Implementierung in Hardware muss folglich die führende Eins beider Operanden gefunden werden. Dies lässt sich durch Schiebeoperationen und einen dekrementierenden Zähler implementieren, welcher bei Erkennung der ersten Eins stoppt. Der finale Zählerwert vor der Ausführung der Multiplikation entspricht somit k_1 bzw. k_2. x_1 und x_2 sind durch die niederwertigeren Bits nach der führenden Eins gegeben und folglich ohne weitere Rechenschritte identifizierbar.

Der durchschnittliche relative Fehler des in ATLAS integrierten Multiplizierers entspricht 3,8 % bei einem maximalen Fehler von 11,11 %. Fehlerbehebungsmethoden, wie unter anderem von Bulic et al. [217] vorgeschlagen, können diese Abweichungen weiter reduzieren. Da der Beschleuniger jedoch nur ganzzahlige Werte verarbeitet, ist der maximale absolute Fehler kleiner als $0,5$. Das Einfügen einer Fehlerkorrekturlogik würde folglich nur zu einer geringen Verbesserung des durchschnittlichen Fehlers führen. Aus diesem Grund verzichtet die Beschleunigerarchitektur von ATLAS auf einen solchen Mechanismus.

Approximierte Aktivierungsfunktion

Abgesehen von den Multiplikationen tragen auch die Aktivierungsfunktionen in der Regel einen nicht vernachlässigbaren Teil zur Latenz der LSTM-Inferenz bei. Darüber hinaus ist die exakte Implementierung dieser Funktionen in Hardware sehr ressourcenintensiv. Um diese Probleme zu bewältigen, verfügt ATLAS über eine approximierende Realisierung der Aktivierungsfunktion. Wie nachfolgende Gleichungen zeigen, handelt es sich bei den Aktivierungsfunktionen der LSTM-Zellen um nicht-lineare Funktionen:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{6.4}$$

$$\sigma(x) = \frac{1}{e^{-x} + 1} \tag{6.5}$$

Für eine ressourcen- und energieeffiziente Implementierung in Hardware, sind diese in der gegebenen Form ungeeignet. Durch den Einsatz von Approximationen lässt sich auch hier eine effiziente Implementierung ableiten. Im Falle der Architektur von ATLAS kann dies in zwei Schritten erfolgen. Hierfür wird Gleichung (6.4) zunächst wie folgt formuliert

$$\tanh(x) = \frac{e^x - e^{-x} + e^{-x} - e^{-x}}{e^x + e^{-x}} = 1 - \frac{2e^{-x}}{e^x + e^{-x}}. \tag{6.6}$$

Durch Division mit e^{-x} lässt sich der hyperbolische Tangens wie folgt mathematisch beschrieben

$$\tanh(x) = 1 - \frac{2}{e^{2x} + 1} = 1 - 2\sigma(-2x) \tag{6.7}$$

$$\sigma(x) = \frac{\tanh(x/2) + 1}{2}. \tag{6.8}$$

Daraus folgt, dass die Sigmoid-Funktion $\sigma(x)$ auch mittels des hyperbolischen Tangens implementiert werden kann. Da die Aktivierungsfunktionen nicht gleichzeitig ausgeführt werden müssen, ist es folglich möglich, nur eine der beiden Aktivierungsfunktionen in der entsprechenden Einheit zu implementieren. Jedoch ist die exakte Implementierung eines hyperbolischen Tangens, wie bereits erwähnt, aufgrund seiner Nichtlinearität ineffizient. Für ATLAS werden daher nachfolgend drei verschiedene Methoden, wie von Namin et al. [218] vorgeschlagen, zur Annäherung an den hyperbolischen Tangens in Hardware implementiert und evaluiert.

Ein solcher Ansatz zur Vermeidung des Rechenaufwands, welcher häufig Anwendung findet, ist die Verwendung von LUTs. Hierdurch ergibt sich allerdings ein größerer Ressourcenverbrauch für die Speicherung der vorberechneten Werte. Dabei können allerdings die folgenden Charakteristika des hyperbolischen Tangens ausgenutzt werden, um die Anzahl der benötigten LUTs zu reduzieren.

$$\tanh(-x) = -\tanh(x) \tag{6.9}$$

$$\lim_{x \to \infty} \tanh(x) = 1 \tag{6.10}$$

Daraus leitet sich ab, dass sich der hyperbolische Tangens asymptotisch den Werten 1 bzw. -1 annähert und punktsymmetrisch zum Ursprung ist. Des Weiteren erreicht die Funktion recht früh einen Wert nahe Eins, weshalb Kumar Meher [219] zeigen konnte, dass die LUT für eine ausreichend gute Approximation nur Werte bis $x = 3$ speichern muss. Für diesen Fall ergibt sich ein maximaler Fehler von $|\tanh(x) - \tanh(3)| < 0.00015$ für $|x| > 3$,

welcher keine nennenswerten Auswirkungen auf die Genauigkeit der DNN-Inferenz hat.

Abgesehen von der Implementierung des hyperbolischen Tangens durch äquidistante Punkte in einer LUT lässt sich die Approximation auch durch mehrere lineare Segmente zur Reduzierung des Fehlers realisieren. Allerdings ist dabei zu beachten, dass diese Approximation nicht den Einsatz eines Multiplizierers erforderlich macht. Für die nachfolgenden Betrachtungen kommt eine Implementierung des hyperbolischen Tangens mittels fünf linearer Segmente zum Einsatz.

$$|\tanh(x)| \approx \begin{cases} x & \text{if } |x| < 0.7 \\ \frac{x}{2} & \text{if } |x| \geq 0.7 \text{ and } |x| < 1.5 \\ 1 & \text{if } |x| \geq 1.5 \end{cases} \tag{6.11}$$

Da die Division mit zwei als Divisor einfach in Hardware zu implementieren ist, lässt sich diese Approximation demnach mit geringem Ressourcenverbrauch in ATLAS integrieren. Die Aktivierungsfunktionseinheit besteht folglich nur aus einem Multiplexer, einigen Komparatoren, um den Eingang des Multiplexers entsprechend auszuwählen, und einer Rechtsschiebeeinheit für die Division.

Abschließend ist auch eine Kombination der beiden Ansätze für die Approximation des hyperbolischen Tangens denkbar. Im Folgenden als hybride Approximation bezeichnet. Ausgehend von den Eigenschaften der Aktivierungsfunktion erfolgt bei der Implementierung dieser Methodik in ATLAS eine lineare Approximation $\tanh(x) \approx x$ für $|x| < 0.7$ und eine LUT-basierte Approximation für $|x| \geq 0.7$. Grund für diese Aufteilung ist der große Fehler der LUTs aufgrund der hohen Steigung des hyperbolischen Tangens für Werte im Bereich $|x| < 0.7$. Das Ersetzen dieses Teils durch lineare Approximation ermöglicht somit eine deutliche Verringerung des durchschnittlichen Fehlers, ohne dass ein erheblicher Hardware-Overhead entsteht.

6.3.3 Systemintegration

Um die Auswirkungen der vorgeschlagenen Hardware-Architektur auf Systemebene zu analysieren, muss ATLAS modelliert und in eine eingebettete Computerplattform integriert werden. Die Modellierung ist für die Abschätzung der Genauigkeit bei Ausführung der Inferenz auf der entworfenen Hardware

sowie der Untersuchung verschiedener Trainingsstrategien notwendig. Zusätzlich ermöglicht die Integration von ATLAS in eine vorhandene Plattform die Analyse des Ressourcenverbrauchs sowie des kritischen Pfads im System, der die maximale mögliche Taktfrequenz vorgibt.

Quantisierung

Die Ausführung von Fließkomma-Operationen auf Hardware ist in Bezug auf Fläche, Latenzzeit und Energiebedarf besonders aufwendig. Daher umfasst die Optimierung des Hardware-Beschleunigers auf Systemebene auch die Quantisierung von Eingaben und Gewichten, was sich negativ auf die Genauigkeit des DNNs auswirkt. Insbesondere die Verwendung von PTQ führt in der Regel zu einem signifikanten Einbruch der Genauigkeit des Modells [35]. Dies ergibt sich im Wesentlichen aus den großen Clipping Ranges, welche zu einer geringen Auflösung bzw. hohe Stufen bei der Quantisierung führen. Dabei leiden speziell kleine DNNs unter diesen Effekten. Infolgedessen empfiehlt sich die Verwendung von QAT zur Analyse des Einflusses von Quantisierung und Anwendung von Approximate Computing.

Zur Modellierung wurde das QKeras Framework [220] verwendet, welches aufgrund seines modularen flexiblen Aufbaus gut für die Analysen geeignet ist. So erlaubt QKeras nicht nur die individuelle Auswahl der Quantisierungen von Eingängen und Gewichten, sondern auch die Implementierung und Integration eigener Aktivierungsfunktionen in das DNN. Im Falle des approximativen Multiplizierers muss dagegen das bestehende LSTM-Layer durch eine eigene Variante ersetzt werden, um auch die Einflüsse dieser Komponente während der Inferenz zu berücksichtigen.

NeoRISC-V Platform

Die realistische Evaluation der Architektur von ATLAS erfordert die Betrachtung des Beschleunigers im Kontext eines vollständigen Systems. Dies ist darauf zurückzuführen, dass die Latenz wesentlich von der Zeit zum Lesen und Schreiben von Daten aus bzw. in den Speicher abhängt. Für die nachfolgenden Untersuchungen wurde daher die NeoRISC-V-Plattform als Grundlage der Systemarchitektur angenommen [221], mit der ATLAS interagiert. Der NeoRISC-V besteht aus einem zweistufigen 32-Bit-RISC-V-Prozessor mit verschiedenen Peripheriegeräten, die zur Entwicklungszeit individuell aktiviert werden können. Er bietet daher ein hohes Maß an Konfigurierbarkeit und

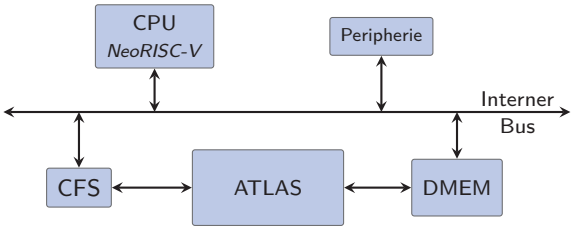


Abbildung 6.6: Integration des LSTM-Beschleunigers in die NeoRISC-V-Plattform

Erweiterbarkeit, was die Integration eigener Hardwaremodule erleichtert. Die Anbindung von Peripheriegeräten und Koprozessoren zur Plattform kann dabei über mehrere Schnittstellen erfolgen. Für die Integration von ATLAS in die Plattform wurde das Custom Function Subsystem (CFS) verwendet, da es eine Ausführung der Inferenz ohne Anhalten des Prozessors ermöglicht. Der Systemaufbau bestehend aus der NeoRISC-V-Plattform und ATLAS ist in Abbildung 6.6 schematisch dargestellt. Das CFS verwendet eine Memory-Mapped-Schnittstelle zur CPU mit 32 direkt zugänglichen 32-Bit-Registern. Im Falle der beispielhaften Integration in die Plattform entspricht das erste Register einem Kontrollregister für die Konfiguration sowie Aktivierung von ATLAS und das zweite einem Statusregister, mit dem die CPU überprüfen kann, ob der Beschleuniger beschäftigt oder im Leerlauf ist.

Innerhalb der Plattform speichert das Data Memory (DMEM)-Modul die Daten für die LSTM-Inferenz. Die enorme Datenmenge, die während der Inferenz zwischen dem Speicher und ATLAS generiert wird, führt zu einer beträchtlichen Belastung des internen Bussystems der Plattform. Dies hat zur Konsequenz, dass sämtliche andere Komponenten zum Stillstand kommen, sobald die Daten an eines der Peripheriegeräte oder gar an den Speicher gesendet oder von diesem gelesen werden müssen. Aus diesem Grund verfügt ATLAS über eine separate Schnittstelle zum DMEM. Dieser Ansatz erhöht nicht nur die verfügbare Speicherbandbreite des Beschleunigers, sondern sorgt folglich auch für eine bessere Performance des Gesamtsystems.

6.3.4 Evaluation

Für die Analyse der Architektur ist die Betrachtung von Metriken erforderlich, welche die wesentlichen Herausforderungen der Ausführung von KI in

IoT-Anwendungen abbilden. Dabei gilt es, Abwägungen hinsichtlich Ressourcenverbrauch, Performance und Genauigkeit zu treffen. Insbesondere die Quantisierung und das Approximate Computing hat dabei einen entscheidenden Einfluss auf die genannten Metriken.

Die nachfolgenden Evaluationen betrachten die Architektur von ATLAS für verschiedene Systemkonfigurationen, um Empfehlungen bezogen auf den Einsatz von approximativen Komponenten bei der DNN-Inferenz abzuleiten. Als beispielhafte Anwendung kommt an dieser Stelle erneut das in Abschnitt 6.2 beschriebene DNN von Wehbi et al. [198] für die Handschrifterkennung zum Einsatz. Ebenso gehen nachfolgende Untersuchungen von dem ebenfalls in Abschnitt 6.2 skizzierten System zur Datenerfassung mittels einer IMU, einem Magnetometer, einem Kraftsensor sowie einem weiteren Beschleuniger am hinteren Ende des Stifts aus.

Das Training der 731 853 Parameter erfolgte mit einem Datensatz aus 49 121 Sequenzen von Wörtern und einzelnen Zeichen, die Evaluationen mit einem Testdatensatz bestehend aus 756 Wörtern. Dabei basieren alle untersuchten Konfigurationen auf einem Ausgangsmodel des DNNs, welches ein Training über 30 Epochen mit einem Adam-Optimierer [222] bei einer Lernrate von 0,01 durchlief. Für die daraus abgeleiteten Modelle wurde jeweils PTQ oder QAT über eine zusätzliche Epoche ausgeführt.

Da es sich bei ATLAS um einen LSTM-Beschleuniger handelt, welcher approximative Operationen für diese Layer nutzt, betrachten nachfolgende Analysen lediglich diese Layer in Bezug auf die relevanten Metriken. Dennoch erfolgte eine Quantisierung der Eingaben und Gewichte der vorherigen Layer des DNNs, sodass auch deren Einfluss auf die Genauigkeit und den Ressourcenverbrauch Berücksichtigung in der Auswertung findet.

Ressourcenverbrauch

Um eine Vergleichbarkeit hinsichtlich des Ressourcenverbrauchs insbesondere mit der Arbeit von Qian et al. [214] herzustellen, gehen die nachfolgenden Untersuchungen von einer Architektur basierend auf 16-Bit-Festkommazahlen mit acht Nachkommastellen aus. Dagegen verwendet Eciton [213] dynamische 8-Bit-Festkommazahlen für die Ein- und Ausgänge. Ein Vergleich des Ressourcenverbrauchs zwischen diesen Architekturen ist in Tabelle 6.2 gegeben.

Insgesamt ergibt die Synthese von ATLAS mit approximativen Multiplizierer und einer LUT-basierten Implementierung der Aktivierungsfunktion mit

Architektur	LUTs	FFs	DSPs	LUTRAM	BRAM
Eciton [213]	4.987 LCs		6	-	22
Qian et al. [214]	1.435	0	8	60	2
ATLAS (LUT 13)	173	67	1	0	1
Approx. Multiplizierer	62	8	0	0	0
Akt.-Funktion					
LUT 9	35	17	0	0	0
LUT 13	55	17	0	0	0
5 Segmente	163	17	0	0	0
Hybrid	107	17	0	0	0

Tabelle 6.2: Ressourcenverbrauch von ATLAS im Vergleich zu anderen aktuellen Beschleuniger-Architekturen

13 äquidistanten Punkten für ein ZedBoard mittels AMD Vivado (Version 2022.1) einen Ressourcenaufwand von 173 LUTs, 67 FFs, einem DSP, und einem BRAM. Im Vergleich zu anderen existierenden Architekturen ist dieser folglich sehr gering. Grund dafür sind im Wesentlichen die folgenden drei Merkmale der Implementierung des Beschleunigers. Erstens sorgt die Wiederverwendung von Komponenten für verschiedene Berechnungen dafür, dass die Architektur von ATLAS kein Modul doppelt implementiert. Darüber hinaus führt die Gestaltung des approximativen Multiplizierers zu einer signifikanten Reduzierung der benötigten Zahl an DSPs im Vergleich zu den beiden anderen Beschleunigern. Außerdem wird durch die Art der Implementierung der Aktivierungsfunktion als kleine LUT ein wichtiger Beitrag zur Minderung des Ressourcenverbrauchs geleistet. Im Gegensatz zu der Realisierung in ATLAS nutzt die von Qian et al. vorgeschlagene Architektur eine LUT mit 256 Elementen, was offenkundig zu einem deutlich höheren Ressourcenverbrauch führt.

Der approximative Multiplizierer benötigt gemäß Tabelle 6.2 62 LUTs und 8 FFs auf dem FPGA. Dabei ist insbesondere hervorzuheben, dass kein Bedarf an DSPs für die Multiplikationen besteht. So erlaubt diese Architektur eine effiziente Implementierung des Multiplizierers im FPGA, da die DSPs in der Regel nicht auf quantisierte Operanden optimiert sind.

Zusätzlich zeigt Tabelle 6.2 den Vergleich des Ressourcenverbrauchs zwischen den verschiedenen Realisierungen der Aktivierungsfunktionen. Die Approximation mittels fünf linearer Segmente liefert zwar die genauesten Ergebnisse, allerdings ist hierfür die Verwendung einer größeren Anzahl an LUTs erforder-

lich als bei allen anderen Varianten. LUT-basierte Implementierungen weisen demgegenüber einen geringen Ressourcenverbrauch auf, jedoch ist hier der durchschnittliche Fehler am größten. Die hybride Approximation bietet folglich einen guten Kompromiss in Bezug auf die Hardwareanforderungen sowie den durchschnittlichen Berechnungsfehler.

Latenz

Für den Einsatz im Digipen zur Handschrifterkennung stellt die Latenz keine besonders wichtige Metrik dar. Dennoch muss sichergestellt werden, dass die Inferenz innerhalb von 80 ms abgeschlossen ist, um einem Datenverlust vorzubeugen. Die Anzahl der benötigten Taktzyklen zur Berechnung eines einzelnen Hidden States im LSTM lässt sich auf Basis theoretischer Überlegungen ableiten. Jedes LSTM-Gate verarbeitet 128 Eingangsaktivierungen, 64 Hidden States und ein Bias. Da Gewichte in den Beschleuniger während laufender Berechnungen geladen werden, ergibt sich für jede Operation eine Latenz von einem Taktzyklus, weshalb die Berechnung eines einzelnen Hidden States folglich 772 Taktzyklen beansprucht. Allerdings muss zusätzlich noch die Zeit für die Konfiguration der Multiplexer, sowie für den Zugriff auf die Daten im BRAM und das Inkrementieren der DMEM-Leseadresse einbezogen werden. Daraus ergeben sich letztlich 775 Taktzyklen für einen einzelnen und demzufolge insgesamt 49 600 Taktzyklen, die ATLAS für die Inferenz der 64 Hidden States des LSTM-Layers benötigt.

Wenig überraschend fällt dieser Wert, insbesondere im Vergleich zur reinen Software-basierten Ausführung des LSTMs, signifikant geringer aus. Für die Ausführung des Layers mittels eine RISC-V ISA ergibt die Simulation in Spike [203] insgesamt 305 862 Instruktionen. Ausgehend von der durchschnittlichen Anzahl an Taktzyklen pro Instruktionen von mindestens 3,54, basierend auf dem CoreMark CPU Benchmark [221], benötigt der NeoRISC-V somit ohne Verwendung des Beschleunigers 1 082 751 Taktzyklen für die LSTM-Inferenz.

Allerdings ist eine theoretische Betrachtung der Latenz in der Regel ungenau. Daher ist zusätzlich die Ermittlung der Latenz für die synthetisierte Hardware, d.h. für den NeoRISC-V mit ATLAS als Koprozessor, erforderlich. Das System wurde bei einer maximalen Taktfrequenz von 67 MHz betrieben, die der kritische Pfad vom Datenspeicher in den approximativen Multiplizierer limitiert. Die dabei gemessene Latenz von 744 µs kommt dabei der theoretischen Latenz von 740 µs bei dieser Taktfrequenz recht nah. Ein fairer Vergleich mit anderen Architekturen ist in dieser Stelle nicht möglich, da meist ähnlich zu [214]

auch die Zahl der Instruktionen für die Berechnung der Fully-Connected Layer in die Evaluation der Latenz einfließt.

Genauigkeit

Die vorherigen Untersuchungen zeigen, dass ATLAS bezogen auf Ressourcenverbrauch und Latenz den Anforderungen von IoT-Anwendungen entspricht. Allerdings ergibt der Einsatz des Hardwarebeschleunigers in solchen Systemen nur Sinn, wenn dieser auch funktionale Vorgaben einhält, d.h. die Genauigkeit des DNNs nicht zu sehr abfällt. Im Folgenden wird diese Metrik daher unter Verwendung von QKeras und den Approximationen der Aktivierungsfunktion sowie der Multiplizierer untersucht.

Als Referenz dient jeweils ein Modell ohne die Verwendung des approximierten hyperbolischen Tangens. Zusätzlich sei an dieser Stelle angemerkt, dass der approximierte Multiplizierer für diese Untersuchungen nur bei der Inferenz zum Einsatz kam und daher die Ergebnisse womöglich etwas schlechter ausfallen. Die trainierten Parameter für die Variante mit (Mitchell) und ohne (non-Mitchell) approximierter Multiplikation sind folglich identisch. Tabelle 6.3 gibt eine Übersicht der Evaluationsergebnisse. Dabei entspricht (x, y) dem Format der Fließkommazahl, wobei x die Anzahl der Nachkommastellen und y die Anzahl der Bits insgesamt angibt.

Entsprechend der Erwartungen, weist das Ausgangsmodell die besten Ergebnisse bezogen auf die Genauigkeit auf. Abgesehen davon ergeben sich für alle Varianten mit approximierten Komponenten im Falle von PTQ deutlich verschlechterte CERs. Dabei leiden insbesondere die Architekturen mit LUTs deutlich hinsichtlich der Genauigkeit, wohingegen für den Fall der Realisierung mittels fünf Segmenten oder hybrid die CER nicht ganz so stark zunimmt. Wesentlich bessere Ergebnisse lassen sich mittels Durchführung von QAT erzielen. Insbesondere die Approximation mit fünf Segmenten liefert in allen Quantisierungen ein sehr gutes Ergebnis, welches hinsichtlich der Genauigkeit dem Ausgangsmodell sehr nahe kommt.

Da der approximative Multiplizierer von ATLAS nur ganzzahlige Werte unterstützt, verschiebt dieser zunächst alle Bits der Eingabedaten fünfmal nach links, um auch Multiplikationen über die Nachkommastellen der Festkommazahlen durchzuführen. Nach Abschluss der Operation macht ATLAS dies wieder rückgängig und gibt das Ergebnis im entsprechenden Format der Festkommazahlen aus. Für alle Quantisierungen innerhalb der Evaluation wurde

Quantisierung	Hyperbolischer Tangens Approximation	PTQ		QAT	
		Non-Mitchell	Mitchell	Non-Mitchell	Mitchell
32-bit float	Baseline	11.040 %			
8-Bit fixed (5,8)	keine	**11,829 %**	**13,030 %**	13,030 %	13,631 %
	LUT 9	20,015 %	22,268 %	11,603 %	13,368 %
	LUT 13	20,203 %	20,766 %	12,730 %	13,969 %
	5 Segmente	12,655 %	14,307 %	**10,627 %**	**11,416 %**
	Hybrid	12,467 %	13,744 %	13,068 %	13,556 %
16-Bit fixed (8,16)	keine	**11,866 %**	**13,256 %**	12,655 %	13,106 %
	LUT 9	18,663 %	27,901 %	13,068 %	13,932 %
	LUT 13	19,076 %	26,023 %	12,730 %	13,030 %
	5 Segmente	12,167 %	15,246 %	**10,702 %**	**11,566 %**
	Hybrid	12,092 %	14,007 %	11,754 %	12,542 %
24-Bit fixed (16,24)	keine	11,904 %	**13,256 %**	12,617 %	12,880 %
	LUT 9	19,076 %	27,976 %	15,246 %	17,236 %
	LUT 13	18,213 %	26,474 %	13,481 %	14,683 %
	5 Segmente	12,204 %	15,621 %	**10,702 %**	**11,265 %**
	Hybrid	**11,791 %**	13,932 %	11,566 %	12,054 %

Tabelle 6.3: Auswirkung verschiedener Approximationen und Trainingsstrategien auf die CER

der gleiche Faktor verwendet, welcher der geringsten Anzahl an Nachkommastellen der verwendeten Schemata entspricht (fünf für 8-Bit-Festkommazahl). Auch für den approximativen Multiplizierer ergeben sich deutlich schlechte Ergebnisse bei Realisierung mittels PTQ. In diesem Kontext ist zu erwähnen, dass insbesondere Quantisierungen mit einer Genauigkeit von 16 Bit bzw. 24 Bit eine erhöhte CER aufweisen. Diese Beobachtung lässt sich durch die Annahme eines 8-Bit-Multiplizierers bei der Evaluierung erklären, was zu einer besseren Performance für 8-Bit-Repräsentationen führt. Dennoch lässt sich hier ebenfalls durch Verwendung von QAT eine deutlich bessere CER erreichen.

Insgesamt zeigen die Ergebnisse, dass sich der Einsatz von Approximate Computing für Anwendungen im IoT-Bereich sehr gut eignet, allerdings eine entsprechende Anpassung der DNN-Parameter notwendig ist. Durch QAT, welches nicht nur die Quantisierung, sondern auch die Approximationen berücksichtigt, wird dabei eine hohe Genauigkeit bei geringem Ressourcenverbrauch und guter Performance erzielt.

6.4 Low-Power FPGA Beschleuniger

Um einen möglichst geringen Energiebedarf im IoT-Gerät zu erzielen, werden häufig ASICs eingesetzt. Gleichzeitig ergaben sich in den letzten Jahren immer wieder neue Formen neuronaler Netze und damit einhergehend Änderungen im Datenfluss. Ein Ende dieses Prozesses ist aktuell nicht in Sicht, da die Suche nach neuartigen DNN-Architekturen weiterhin ein relevantes Thema in der Forschung ist. Aus diesem Grund ist der Einsatz von ASICs häufig nicht sinnvoll, da diese nur eine geringe Flexibilität bieten. Als Alternative bieten sich daher FPGAs an, welche neben der Rekonfigurierbarkeit zusätzlich geringere Entwicklungskosten der Anwendung auf dem FPGA bieten. Nachfolgende Untersuchungen zielen darauf ab, Low-Power FPGAs auf den Einsatz in IoT-Anwendungen zu verwenden und insbesondere den resultierenden Energiebedarf des Gesamtsystems zu evaluieren. Die Erläuterungen und Ergebnisse wurden vom Autor in [Kre24c] veröffentlicht.

In den letzten Jahren haben sich TCNs als beste Form von RNNs für die Verarbeitungen von Zeitseriendaten in IoT-Anwendungen herausgestellt [223]. Dabei verfügen diese insbesondere im Vergleich zu LSTMs über eine geringere Rechenkomplexität, weshalb ein solches TCN den Ausgangspunkt der Entwicklung einer Hardwarearchitektur für den FPGA darstellt. Der TCN-Beschleuniger LOTTA ist derart entworfen, dass er die auf dem FPGA vorhandenen Ressourcen möglichst gut ausnutzt und als Co-Prozessor agieren kann. Aufgrund des stark limitierten Speicherplatzes auf dem FPGA verfügt LOTTA des Weiteren über Quad Serial Peripheral Interface (SPI) zur Anbindung externer Speicher an den Beschleuniger. Diese Architektur erlaubt folglich die Inferenz größerer TCNs auf dem FPGA.

6.4.1 Verwandte Arbeiten

Die Inferenz von TCNs erfordert eine hohe Anzahl an MAC-Operationen, weshalb deren Optimierung auf Hard- und Software-Ebene zu einer deutlich verbesserten Energieeffizienz und Performance führt. Beispielhaft wurde von Serdyuk et al. [Ser23] nachgewiesen, dass Approximate Computing und Quantisierung eines vortrainierten TCNs zu geringeren Hardwareanforderungen und einer geringeren Modellgröße führen. Darüber hinaus demonstrieren Ingolfsson et al. [224] für ein kleines TCN, wie verschiedene Softwareoptimierungen die Energieeffizienz der Inferenz auf Mikrocontroller bei Beibehaltung der Genauigkeit des Modells deutlich verbessert. Allerdings haben diese Hardwareplattformen den Nachteil, dass diese nicht für die Ausführung

eines TCNs optimiert sind und daher in der Regel keine hohe Energieeffizienz erzielen.

Daher wurden bereits spezialisierte Hardwarebeschleuniger für die Inferenz von TCNs vorgeschlagen. Auch für die Implementierung auf einem FPGA wurden verschiedene Architekturen präsentiert [225, 226], welche sich die vorhandenen DSPs zunutze machen. Allerdings gehen diese Arbeiten von FPGAs mit vielen verfügbaren Hardwareressourcen aus, welche die Anforderungen von IoT-Anwendungen nicht erfüllen. Folglich fehlt es im aktuellen Stand der Technik an Lösungen, wie Low-Power FPGAs für die Inferenz von TCNs in solchen Systemen mit strikten Einschränkungen des Energiebedarfs eingesetzt werden könnten.

Im Gegensatz dazu finden sich verschiedene Architekturen für ASICs in der Literatur. So präsentierten Bernardo et al. [227] den Beschleuniger UltraTrail, welcher für den Einsatz eines TCNs zur Erkennung von Schlüsselwörtern optimiert ist. Um eine möglichst hohe Performance zu erzielen, setzt diese Architektur auf ein MAC-Array, welches die parallele Verarbeitung von Daten energieeffizient ermöglicht und gleichzeitig einen hohen Datendurchsatz bietet. In einer sehr ähnlichen Form wurde der Beschleuniger SOMA [228] entworfen. Problematisch an beiden Architekturen ist deren kleiner integrierter Speicher, wodurch die Anwendbarkeit der Beschleuniger auf andere Einsatzgebiete stark limitiert ist. Zudem ist der Aufbau bestehend aus 64 MAC-Einheiten nicht mit den verfügbaren Hardwareressourcen eines Low-Power FPGAs vereinbar. Folglich lässt sich konstatieren, dass gegenwärtig keine Architektur für einen TCN-Beschleuniger existiert, die sich ohne Modifikationen auf einen kleinen FPGA übertragen ließe.

6.4.2 Hardwarearchitektur

Grundsätzlich existieren zwei verschiedene Möglichkeiten für die Implementierung der Inferenz eines TCNs in Hardware. So kann die Architektur entweder speicher- oder rechenintensiv ausgelegt werden. Letzteres bedeutet, dass alle Berechnungen für die Inferenz in jedem Zeitschritt stattfinden und somit keine Sicherung von Zwischenergebnissen erfolgt. Diese Methodik ist insbesondere für leistungsstarke Hardwareplattformen geeignet. Für IoT-Plattformen ist dieser Ansatz allerdings von Nachteil, da bei kleinen Beschleunigern die Rechenleistung stark eingeschränkt ist. Aus diesem Grund wurde die Architektur von LOTTA speicherintensiv ausgelegt, wobei der Beschleuniger Zwischenergebnisse abspeichert und im nächsten Zeitschritt wiederver-

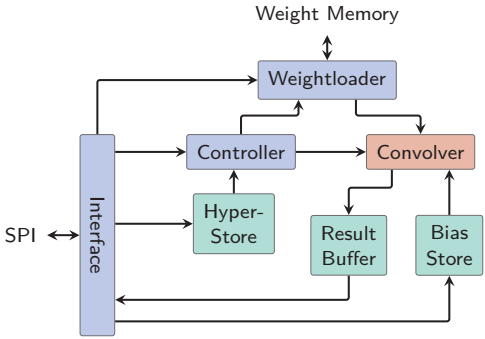


Abbildung 6.7: LOTTA-Architektur bestehend aus Kontrolllogik (blau), Speichern (grün), arithmetischer Einheit (rot) und externen Schnittstellen

wendet. Mit diesem Ansatz erfolgt in einem kausalen TCN folglich nur die Berechnung der Aktivierungen des letzten Zeitschritts.

Basierend auf diesem Datenfluss lässt sich so eine passende Architektur des Hardwarebeschleunigers ableiten. Eine Übersicht des Designs von LOTTA ist in Abbildung 6.7 gegeben. Damit LOTTA die TCN-Inferenz performant und möglichst energieeffizient auf dem FPGA ausführen kann, sollten zunächst alle Hardwareressourcen zur Verfügung stehen. Daher ist LOTTA als Co-Prozessor für einen Mikrocontroller ausgelegt, welcher sich über SPI ansteuern lässt. Für die Ausführung der Inferenz überträgt der Mikrocontroller unter anderem die TCN-Konfiguration, welche der *Controller* entsprechend verarbeitet. Dieser übernimmt das Speichermanagement und sorgt dafür, dass der *Convolver*, die arithmetische Einheit des Beschleunigers, die korrekten Daten für die anstehenden Berechnungen erhält. Der *Convolver* verfügt als reine Recheneinheit über keinerlei Zustand und verarbeitet die Daten daher wie vom *Controller* vorgegeben. Zuletzt ist außerdem auch der *Weightloader* als wichtige Komponente hervorzuheben, welcher die Gewichte aus dem externen Speicher in den *Convolver* zur Berechnung lädt. Nachfolgend wird der Aufbau ebenjener Module von LOTTA näher erläutert.

Convolver

Wie bereits erwähnt, besteht die wesentliche Aufgabe des *Convolvers* in der Ausführung der arithmetischen Operationen für die TCN-Inferenz. Daher soll zunächst nochmals kurz auf die Struktur eines solchen RNNs eingegangen

werden. Ein TCN besteht aus mehreren Schichten, welche jeweils eine gewisse Anzahl an Faltungen abhängig von der Menge an Filter durchführen. Dabei legen die Filterbreite sowie die jeweilige Dilatation die einzubeziehenden Zeitschritte jeder Schicht fest. Mittels der Dilatation lässt sich einfach und schnell ein großes rezeptives Feld erzeugen, was für viele Zeitserien-basierte Anwendungen von Vorteil ist. Eine solche Faltungsoperation ist wie folgt definiert:

Definition 7 (Dilatierte Faltung)**.** *Eine dilatierte Faltung F auf einem Element s mit Filter f_m für eine eindimensionale Eingangssequenz x und einem Bias b_m ist gegeben durch*

$$F(s,m) = \sum_{i=0}^{k-1} \sum_{j=0}^{c-1} f_m(i,j) \cdot x_{s,j-i\cdot d} + b_m,$$

mit der Kernelgröße k, der Anzahl an Kanälen c und der Dilatation d.

Definition 7 zeigt deutlich, dass MAC-Operationen den Großteil aller Berechnungen ausmachen und folglich eine Beschleunigung dieser den größten Leistungszuwachs ermöglichen. Für eine möglichst hohe Energieeffizienz nutzt LOTTA die auf dem FPGA vorhandenen DSPs, woraus gleichzeitig eine geringe Latenz resultiert. Sobald die Berechnungen eines Layers nicht vollständig auf die vorhandenen MAC-Einheiten im *Convolver* abgebildet werden können, muss der *Controller* in diesen Fällen mittels Zero-Padding eine Verfälschung der Ergebnisse verhindern.

Controller

Im Zentrum der Architektur von LOTTA steht der *Controller*, welcher über jeweils eine Verbindung zu allen wesentlichen Komponenten des Beschleunigers verfügt. Darüber hinaus nutzt er zwei interne Speicherblöcke zum Sichern der Zwischenergebnisse sowie dem Speichern der Lese- und Schreibpointer. Die Steuerung der Inferenz erfolgt über eine FSM, welche schematisch in Abbildung 6.8 dargestellt ist.

Dabei sorgt die Initialisierung (Init) zunächst für ein Löschen aller Daten aus den Speichern, um Interferenz zwischen zwei Durchläufen zu verhindern. Anschließend lädt der *Controller* in der Konfigurationsphase (Config) die Netzwerkparameter des auszuführenden TCNs aus dem angeschlossenen *Hyperstore*. Dieser Speicher wiederum ist direkt über SPI von außen adressierbar, wodurch LOTTA eine hohe Konfigurierbarkeit erreicht. So ermöglicht diese

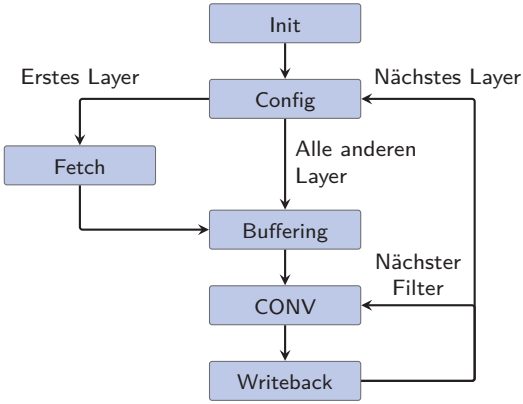


Abbildung 6.8: FSM des Controllers

externe Schnittstelle die Ausführung von TCNs unterschiedlicher Größe auf dem Beschleuniger, sodass LOTTA auch zur Laufzeit verschiedene Aufgaben ausführen kann.

Vor der Durchführung der notwendigen Berechnungen sind zwei weitere Schritte notwendig. Handelt es sich um das erste Layer des TCNs, so wird zunächst in den Fetch-Zustand gewechselt, in welchem die FSM die Übertragung von Daten aus dem externen Speicher initiiert und darin verbleibt bis der interne Puffer ausreichend Daten für den Start der Inferenz zwischengespeichert hat. Damit können die Berechnungen allerdings noch immer nicht beginnen. Um eine parallele Ausführung der MAC-Operationen in unterschiedlichen DSPs zu ermöglichen, überträgt der Beschleuniger im Buffering-Zustand die Eingangsdaten oder Zwischenergebnisse aus einem großen Single-Port-RAM in einen kleinen dedizierten Speicher. Da die MAC-Einheiten nicht einem festen Kanal oder einem Filter zugeordnet sind und die Gewichte sequenziell verarbeitet werden, verhindert diese Architektur ein wiederholtes Laden derselben Parameter aus dem externen Speicher. Dies resultiert in einer gesteigerten Performance sowie Energieeffizienz.

Nachdem die dedizierten Speicherblöcke die benötigten Daten erhalten haben, wechselt der *Controller* in den CONV-Zustand und startet die Berechnungen im *Convolver* durch das Übertragen der benötigten Daten. Sobald alle MAC-Operationen durchgeführt wurden, erfolgt die Sicherung der Zwischenergebnisse zurück in den Single-Port RAM im Writeback-Zustand. Ist die Inferenz des Layers abgeschlossen, wechselt der *Controller* zurück in

den Config-Zustand, um die Parameter des nächsten Layers zu laden. Andernfalls geht der *Controller* zurück in den CONV-Zustand und startet die nachfolgenden Berechnungen des nächsten Filters.

Weightloader

Bei den meisten KI-Anwendungen stellt das Laden von Daten aus dem externen Speicher den größten Engpass im System dar. Dies ist auch für LOTTA der Fall, da angenommen werden muss, dass der interne Speicher des FPGAs nicht für alle Gewichte eines TCNs ausreicht. Eine hohe Bandbreite zum Speicher ist zudem erforderlich, um die parallelen MAC-Einheiten im *Convolver* ausreichend schnell mit neuen Daten zu versorgen. Die Architektur des *Weightloader* von LOTTA, welcher hierfür zuständig ist, löst dieses Problem durch die Kombination zweier gängiger Techniken.

Zum einen nutzt LOTTA ein skalierbares SPI im *Weightloader*, wodurch mehrere externe Speicher parallel an den Beschleuniger angeschlossen werden können. Dies erlaubt die Verwendung mehrerer Speicher-Module zum Laden der Daten und damit eine signifikante Erhöhung der verfügbaren Bandbreite. Zusätzlich werden durch Quad SPI vier Bits pro Takt und integrierte Schaltkreise (ICs) an LOTTA übertragen.

Darüber hinaus verwendet LOTTA eine höhere Taktfrequenz für die Schnittstelle zu den externen Speichern, da die Architektur eine Entkopplung des Ladens der Daten von den im Convolver stattfindenden Berechnungen erlaubt. Für die domänenübergreifende Taktung setzt die Architektur von LOTTA auf asynchrone Puffer, die einen unkomplizierten Datenaustausch zwischen den beiden Taktdomänen ermöglichen.

SPI Slave Interface

Für die Kommunikation zu einem Master-Gerät ist das Slave-Interface von LOTTA essenziell. Hierüber findet nicht nur die Konfiguration des Beschleunigers statt, sondern auch die Übertragung von Gewichten, Biases, Eingangsdaten und den finalen Ergebnissen. Da LOTTA als Co-Prozessor für einen gewöhnlichen Mikrocontroller arbeiten soll, ist die Schnittstelle mittels SPI realisiert, wobei es sich um einen weit verbreiteten Standard handelt. Somit wird sichergestellt, dass viele Mikrocontroller-Plattformen auf LOTTA zugreifen können.

Wie bereits erwähnt erfolgt die Konfiguration über den *Hyperstore*, welcher die Hyperparameter des TCNs zur Laufzeit speichert. Nach dem Schreiben der Daten von außen in diesen Speicher erfolgt ein automatischer Neustart der Inferenz. Davor müssen allerdings zunächst noch die Gewichte und Biases in den entsprechenden Speichern abgelegt werden. Dieser Vorgang erfolgt ebenso über SPI. Währenddessen leitet das Interface die eingehenden Gewichte automatisch zum *Weightloader* weiter, welcher diese entsprechend an die externen Speicher überträgt. Für die Biases ist dagegen genug Speicher auf dem FPGA vorhanden, weshalb in diesem Fall die empfangenen Daten direkt in den dafür ausgelegten *Biasstore* geladen werden. Ebenso empfängt LOTTA die Eingangsdaten von Sensoren oder anderen externen Quellen über SPI und leitet diese, wie bereits zuvor beschrieben, intern über den *Controller* an den *Convolver* weiter. Nach Abschluss der Inferenz lädt der externe Mikrocontroller die Ergebnisse der Berechnungen schließlich über diese Schnittstelle für die weitere Verarbeitung. Dafür verfügt LOTTA über den *Resultbuffer*, welcher unabhängig von den anderen Komponenten des Beschleunigers von außen adressiert und ausgelesen wird.

6.4.3 Evaluation

Für eine realitätsnahe Evaluation von LOTTA bedarf es einer entsprechenden FPGA-Plattform, welche auf eine geringe Leistungsaufnahme und damit auf den Einsatz in IoT abzielt. Dabei ist allerdings zu beachten, dass der FPGA genügend Hardwareressourcen zur Verfügung stellt, um eine solche Architektur in Hardware implementieren zu können. Eine der wenigen Plattformen, welche diese Anforderungen erfüllt, ist der Lattice iCE40 UP5K FPGA. Sie verfügt über 5.280 Logic Cells (LCs) und acht DSPs für die Realisierung der notwendigen Logik zur Beschleunigung von TCNs sowie über 1 Mbit internen eingebetteten Speicher. Damit ist der FPGA sehr gut für die Architektur von LOTTA geeignet und soll daher als Plattform für die weiteren Evaluationen sowie die anschließenden Fallstudien dienen.

Dennoch ist der Speicherplatz nicht ausreichend, um in jedem Fall alle Gewichte des TCNs neben den Biases und den Aktivierungen zu speichern. Aus diesem Grund erweitern zusätzliche Speicher-ICs die Plattform, welche das externe Interface des *Weightloaders* für den Datenaustausch nutzen. Akkubetriebene IoT-Anwendungen, wie beispielsweise dem Digipen zur Handschrifterkennung, sind der Gefahr einer Unterbrechung der Versorgungsspannung ausgesetzt. Um das Laden der Gewichte in den Speicher über BLE oder eine andere drahtlose Schnittstelle zu vermeiden, sollte daher in diesen Fällen auf

	Khatwani et al. [229]	Eciton [213]	LOTTA
Plattform	Artix-7	iCE40	iCE40
DNN	CNN	LSTM	TCN
LCs	4.635	4.987	3.990
DSPs	40	6	4
BRAM	204	22	13
SPRAM	-	4	4

Tabelle 6.4: Ressourcenverbrauch von LOTTA auf einem Lattice iCE40 UP5K FPGA

nichtflüchtige Speicher zurückgegriffen werden. Aus diesem Grund besteht der Testaufbau neben dem FPGA aus vier Non-Volatile Static Random-Access Memory (NVSRAM) Modulen aus der Infineon CY14V101 Reihe mit 1 Mbit Speicher pro IC. Damit bleiben die Gewichte auch nach einem Ausfall der Spannungsversorgung im Speicher erhalten. Jeder dieser NVSRAMs ist via Quand SPI an den *Weightloader* angebunden, sodass insgesamt 16 Bits parallel übertragen werden. Zusammengefasst ergeben sich somit 1 Mbit an eingebettetem Speicher im FPGA für die Aktivierungen sowie 4 Mbit externer Speicher für die Gewichte des TCNs.

Ressourcenverbrauch

Eine wesentliche nicht-funktionale Metrik für IoT-Anwendungen und speziell im Kontext des eingesetzten Low-Power-FPGAs ist der Ressourcenverbrauch. Um LOTTA auf dieser Plattform einzusetzen, sollte die Architektur folglich einen möglichst geringen Hardwareaufwand aufweisen. Dies hat in der Regel gleichzeitig positive Auswirkungen auf den Energiebedarf des Systems, da die dynamische Verlustleistung durch weniger verwendete Logikelemente geringer ausfällt. Für die nachfolgenden Analysen wurde LOTTA als Beschleuniger für 8-Bit Integer Gewichte und Aktivierungen auf dem FPGA implementiert. Wie beispielsweise von Serdyuk et al. [Ser23] nachgewiesen, hat dieser Grad der Quantisierung bei Anwendung von QAT meist keine großen Auswirkungen auf die Genauigkeit des TCNs. Eine Übersicht des Ressourcenverbrauchs und ein Vergleich zu vergleichbaren Architekturen aus dem aktuellen Stand der Technik ist in Tabelle 6.4 gegeben.

Zunächst soll der Ressourcenverbrauch von LOTTA selbst genauer analysiert werden. Insgesamt ergibt sich für die Architektur ein Ressourcenverbrauch, der im Rahmen der vorhandenen Komponenten des FPGAs liegt und darüber

hinaus noch Platz für weitere Module bietet. So ließen sich weitere DSPs und LCs beispielsweise für den *Convolver* oder für einen zusätzlichen Hardwarebeschleuniger verwenden. Die Architektur von LOTTA erreicht auf dem FPGA eine Frequenz von 30 MHz für den *Weightloader*, generiert von der internen Phase-Locked Loop (PLL). Der restliche Teil des Beschleunigers wird mit einer vom internen Prescaler generierten Taktfrequenz von 24 MHz ausgeführt.

Im aktuellen Stand der Technik findet sich, wie zuvor erwähnt, kein TCN-Beschleuniger, welcher sich für den Einsatz auf kleinen Low-Power-FPGAs eignet. Folglich ist ein Vergleich mit einer ähnlichen Architektur hier nicht möglich. Für die Abschätzung des relativen Ressourcenverbrauchs zur Beschleunigung eines DNNs auf einem FPGA erfolgt daher ein Vergleich von LOTTA mit zwei anderen aktuellen Low-Power-Architekturen. Bei diesen handelt es sich um Eciton [213], entwickelt für den Einsatz in der prädiktiven Instandhaltung, und dem Beschleuniger von Khatwani et al. [229], optimiert für Geräte zur Erkennung von Artefakten im EKG.

Wie aus Tabelle 6.4 hervorgeht, benötigt LOTTA im Vergleich zu den beiden Architekturen deutlich weniger Hardwareressourcen. Dabei offenbart sich insbesondere auch im Vergleich zu [229], dass die Anforderungen von TCNs an die Hardware zur Verarbeitung von Zeitseriendaten geringfügiger ausfallen als bei der Verwendung von zweidimensionalen Faltungsoperationen. So ergibt sich für die Inferenz des CNNs ein Ressourcenverbrauch von 40 DSPs und über 200 BRAMs auf dem AMD FPGA. Für die Evaluation von Eciton wurde der gleiche FPGA verwendet, wie in dieser Arbeit. Allerdings ist dieser Beschleuniger für LSTMs gedacht, woraus sich andere Anforderungen an die Hardware ableiten. Dennoch lassen sich aus diesem Vergleich zwei Erkenntnisse ableiten. Zum einen ist die Architektur von LOTTA derart effizient, dass sie sich gut für den Einsatz in IoT-Anwendungen eignet. Zum anderen offenbart der Vergleich die zusätzlichen Vorteile von TCNs über LSTMs. So liefern TCNs häufig nicht nur eine bessere Genauigkeit, sondern ermöglichen auch eine Implementierung in Hardware mit geringem Ressourcenverbrauch.

Leistungsaufnahme

Neben der stark eingeschränkt verfügbaren Fläche zur Einbettung von Hardwarebeschleunigern in IoT-Geräten, nimmt auch die Leistungsaufnahme eine entscheidende Rolle ein. So müssen solche Systeme über einen längeren

	FPGA	NVSRAM		Overall
	VCC	VCC	VCCQ	
Spannung	1,2 V	3,3 V	1,8 V	
Strom	5,0 mA	4,6 mA	4,1 mA	
Leistungsaufn.	**6.0 mW**	8,28 mW	13,53 mW	**27.81 mW**

Tabelle 6.5: Messergebnisse der Leistungsaufnahme von LOTTA

Zeitraum (mehrere Stunden bis mehrere Tage oder Wochen) unterbrechungsfrei ohne Aufladen funktionieren, gleichzeitig aber häufig auch klein und handlich sein, weshalb nur eine geringe Akkukapazität zur Verfügung steht. Diese Anwendungsgebiete verlangen folglich den Einsatz von Plattformen mit geringer Leistungsaufnahme.

Aus diesem Grund bedarf es zusätzlich auch einer energieeffizienten FPGA-Plattform mit einer niedrigen statischen Leistungsaufnahme wie dem Lattice iCE40 UP5K FPGA. Für die Evaluation des Energiebedarfs des FPGAs und der vier externen NVSRAMs-Modulen kam ein TCN-Layer bestehend aus 100 Filtern und einer Kernelgröße von 3 zum Einsatz. Dies stellt ein großes Layer für IoT-Anwendungen dar, weshalb es sich bei der Betrachtung der kontinuierlichen Ausführung einer solchen Schicht um ein Worst-Case-Szenario handelt. Folglich werden pausenlos Daten an den *Controller* gesendet, sodass LOTTA Berechnungen ohne Unterbrechung ausführt. Die Leistungsaufnahme des Systems basiert auf der indirekten Messung des Stroms durch Ermittlung der abfallenden Spannung über einen 1 Ω Shunt-Widerstand, welcher jeweils in Serie in die Versorgungsleitungen für FPGA und NVSRAM integriert wurde. Eine detaillierte Übersicht der Messergebnisse der Systemkomponenten ist in Tabelle 6.5 gegeben.

Erwartungsgemäß entfällt ein besonders hoher Anteil des Energiebedarfs auf die externen Speicher. Mit 21,81 mW ist die Leistungsaufnahme etwa 3,6-mal so hoch wie die des Beschleunigers auf dem FPGA, welcher mit 6,0 mW einen geringen Wert erreicht. Daraus lassen sich zwei Aussagen ableiten. Zum einen liegt der Energiebedarf von LOTTA bei einer Implementierung ohne externen Speicher auf einem sehr guten Niveau bezogen auf die Anforderungen von IoT-Anwendungen. Zusätzlich zeigen die Messergebnisse aber auch, dass das Gesamtsystem trotz Integration von externem NVSRAM, welcher für größere TCNs unumgänglich ist, eine geringe Leistungsaufnahme von 27,81 mW erzielt. Der Vergleich mit ähnlichen Architekturen in Tabelle 6.6 beweist dabei, dass der Energiebedarf selbst bei Nutzung externer Speicher gering ausfällt

	Eciton [213]	[229]	[224]		LOTTA
DNN	LSTM	CNN	TCN	TCN	TCN
Plattform	iCE40	Artix-7	GAP8	STM32L475	iCE40
Externer Speicher	✗	✗	✗	✗	✓
Frequenz (MHz)	17	11,1	100	80	24
Power (mW)	17	32	38,52	42,75	27,81
Durchsatz (GOP/s)	0,067	–	0,381	0,018	0,12
Energieeffizienz (GOP/s/W)	3,94	–	9,91	0,43	4,40

Tabelle 6.6: Leistungsaufnahme und Performance von LOTTA im Vergleich zu verwandten Arbeiten

und IoT-Systeme LOTTA in Bezug auf diese Metrik recht problemlos integrieren können.

Der GreenWaves GAP8 [230], welcher explizit für den Einsatz in IoT-Anwendungen entwickelt wurde, nimmt nach Ingolfsson et al. [224] 38,52 mW auf. Ebenfalls ergibt sich für den STM32L475 ein signifikant höherer Energiebedarf von 42,75 mW, obwohl ebenjener auch hinsichtlich der Anforderungen von IoT optimiert wurde. Im Vergleich zu bekannten Plattformen schneidet lediglich der LSTM-Beschleuniger Eciton besser ab, welcher aufgrund der ausschließlichen Nutzung von internem BRAM eine geringere Leistungsaufnahme erzielt. Würde man diese, von Chen et al. [213] gemessenen Werte jedoch der Konfiguration von LOTTA ohne externen Speicher gegenüberstellen, ergäbe sich ein anderes Bild.

Dies belegt zusätzlich die ermittelte Energieeffizienz der beiden Beschleuniger. Hier lässt sich für LOTTA mit 4,40 GOP/s/W ein Gewinn von etwa 11,68 % im Vergleich zu Eciton ermitteln. In dieser Kategorie schneidet lediglich der GreenWaves GAP8 besser als LOTTA ab, was den Vorteil von Low-Power-ASICs gegenüber FPGA-basierten Beschleunigern verdeutlicht. Dabei sollte allerdings auch nicht außer Acht gelassen werden, dass sich die Ergebnisse für LOTTA auf ein System mit externen NVSRAMs beziehen, welche wie in Tabelle 6.5 gezeigt, einen enormen Einfluss auf die Energieeffizienz haben. Gleichzeitig wurde dies für den GAP8 nicht berücksichtigt, da für die Messung ein deutlich kleineres TCN zum Einsatz kam. Abgesehen davon zeigen die Ergebnisse hinsichtlich der Leistungsaufnahme deutlich die gute Eignung von LOTTA für IoT-Anwendungen. Gleichzeitig lässt sich daraus ableiten, dass Low-Power-FPGAs durchaus für die Integration in KI-Systeme mit strikten Anforderungen an eine geringe Leistungsaufnahme geeignet sind.

Latenz

Für die Evaluation der Latenz wurde, wie zuvor, ein TCN-Layer bestehend aus 100 Eingangskanälen, 100 Filtern und einer Kernelgröße von 3 als Referenz ausgewählt. Die Bestimmung einer akkuraten Abschätzung der Latenz erfolgte dabei durch die Analyse von Simulationen des Systems unter Berücksichtigung der zuvor genannten maximalen Taktraten. So lässt sich die Dauer der Inferenz direkt aus den Ergebnissen ablesen. Die Messungen ergaben dabei für den beschriebenen Anwendungsfall eine Performance von 0,12 GOP/s, wodurch LOTTA erneut im Vergleich zu den anderen Plattformen sehr gut abschneidet (s. Tabelle 6.6). So erzielt lediglich der GAP8 mit 0,381 GOP/s einen höheren Wert, welcher allerdings auch in der deutlich höheren Taktfrequenz begründet ist. Folglich beweisen die dargestellten Ergebnisse bezüglich Energieverbrauch und Latenz die grundsätzliche Eignung von LOTTA für den Einsatz in IoT-Anwendungen.

6.4.4 Hardware-orientierte Modell-Optimierung

In der Regel werden neuronale Netze für die Ausführung in der Cloud oder in eingebetteten HPC-Plattformen entwickelt, welche weniger strikten Einschränkungen hinsichtlich Energiebedarf im Vergleich zu IoT-Anwendungen unterliegen. Solche DNNs sind folglich meist nicht für den Einsatz in IoT geeignet, da sie entweder zu viel Speicher benötigen oder zu hohe Anforderungen an die Leistungsfähigkeit des Systems stellen. Vorheriger Abschnitt 6.3 stellte bereits verschiedene Methoden vor, wie solche DNNs mittels Approximate Computing und Quantisierung dennoch in IoT-Anwendungen zum Einsatz kommen können. Als Ergänzung dazu soll nachfolgend eine weitere Möglichkeit näher betrachtet werden: eine Hardware-orientierte Hyperparametersuche für DNNs. Dabei wird eine Grundstruktur des Netzes vorgegeben und lediglich eine Kombination von Hyperparametern der Layer gesucht, die eine möglichst hohe Genauigkeit des DNNs bei möglichst geringer Rechenkomplexität erzielt. Eine schematische Darstellung dieses Ablaufs ist in Abbildung 6.9 gezeigt.

Für die im nachfolgenden Abschnitt durchgeführten Fallstudien kommt als Basis der Hyperparametersuche das Open-Source Framework Optuna [231] zum Einsatz, welches verschiedene Konfigurationen des TCNs evaluiert und im Falle einer Mehrzieloptimierung eine Pareto-Front als Ergebnis ausgibt. Die Einbeziehung der Hardwarespezifikationen des Beschleunigers mitsamt

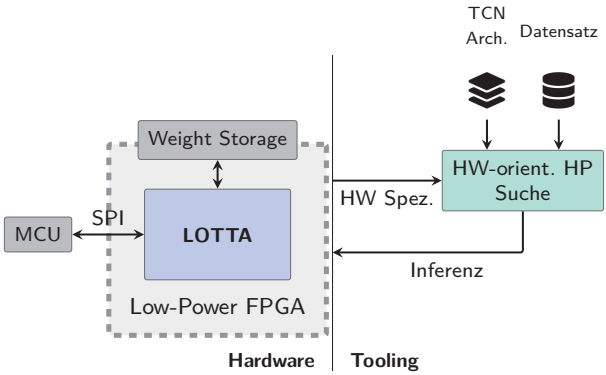


Abbildung 6.9: Hardware-orientierte Hyperparametersuche

externem Speicher erfolgt dabei über die Berechnung des benötigten Speicherplatzes für die Inferenz. Dabei wird, wie von Saha et al. [232] vorgeschlagen, zwischen dem Speicher für die Gewichte und dem für die Zwischenergebnisse unterschieden. Sobald die Speicheranforderungen die vorhandenen Hardwareressourcen von LOTTA überschreiten, wird die Hyperparameterkombination als invalide gekennzeichnet und im weiteren Verlauf der Optimierung nicht weiter berücksichtigt. Optuna versucht durch Mutation der Hyperparameter innerhalb eines vom Benutzer vorgegebenen Rahmens ein optimales TCN für den verwendeten Datensatz zu finden. Die Evaluation erfolgt dabei anhand der Genauigkeit des trainierten Netzes sowie der Anzahl an auszuführenden Operationen für die Inferenz des TCNs. So erfolgt schließlich basierend auf den Anforderungen an das System, wie beispielsweise maximale Latenz oder minimale Genauigkeit, die Auswahl des besten TCNs für die gegebene Anwendung sowie die Bereitstellung der entsprechenden Konfigurationsdaten für die Ausführung des Netzes auf LOTTA.

6.4.5 Fallstudien

Nachfolgende Studien nutzen die vorgestellte Hyperparametersuche für zwei unterschiedliche Anwendungsfälle, um basierend auf den zuvor vorgestellten Evaluationsergebnissen die Eignung von LOTTA für den Einsatz in IoT-Systemen zu beweisen. Diese beiden zeitserienbasierte Aufgaben bringen unterschiedliche Anforderungen mit sich, was zusätzlich die Vielseitigkeit von LOTTA unterstreichen soll.

Als Basis der Hyperparametersuche dient ein RNN bestehend aus einem TCN-Layer gefolgt von einem Max-Pooling mit Kernelgröße 2, einem Flatten-Layer und zwei Fully-Connected Schichten mit 32 bzw. n Ausgangsaktivierungen. Dabei gilt im ersten Anwendungsfall, der Trägheitsnavigation, $n = 2$, für die Sprachsteuerung entspricht dies der Anzahl an zu erkennenden Wörtern, in diesem Fall $n = 12$. Das Training der RNNs nutzt eine Batch-Größe von 400 Zeitschritten. Unter diesen Bedingungen sucht Optuna nach einer optimierten Kombination aus folgenden Hyperparametern für das TCN-Layer, wobei jeder Stack aus mindestens zwei 1D-CONVs bestehen muss:

- Filter: 20 – 100
- Kernelgröße: 1 – 3
- Dilations: (1, 2, 4, 8)
- Stacks: 1 – 5
- Dropout-Rate: 0.0 – 0.5

Trägheitsnavigation

Die Herausforderung bei der Trägheitsnavigation besteht darin, die Bewegungen einer Person ausschließlich mit Hilfe von IMUs zu schätzen. Dabei werden in der Regel Daten von Beschleunigungssensoren, Gyroskopen und Magnetometern verwendet, die jedoch sehr anfällig für Rauschen sind. Aus diesem Grund kommen zunehmend neuronale Netze zum Einsatz, die den Einfluss solcher Störungen in den Sensoren reduzieren. Da allerdings der Energiebedarf bei diesen Anwendungen eine wichtige Rolle spielt, muss die Hardware gut auf die Architektur des DNNs abgestimmt sein, um eine hohe Energieeffizienz zu erreichen. Aus diesem Grund soll nachfolgend untersucht werden, inwiefern LOTTA für den Einsatz in einer solchen Anwendung geeignet ist.

Als Basis für das Training des TCNs dient dabei der RoNIN-Datensatz [233], welcher IMU-Daten von über 42,7 Stunden umfasst und damit laut den Herausgebern aktuell den größten Datensatz für Trägheitsnavigation darstellt. Die Ergebnisse der Suche nach optimalen Hyperparametern für das zuvor beschriebene TCN-Layer sind in Abbildung 6.10 dargestellt. Aus den 55 durchgeführten Evaluationen ergibt sich eine Pareto-Front in Bezug auf Mean Squared Error (MSE) und Rechenkomplexität. Einen guten Kompromiss zwischen diesen Parametern stellt die Konfiguration mit 24 Filtern, Kernelgröße

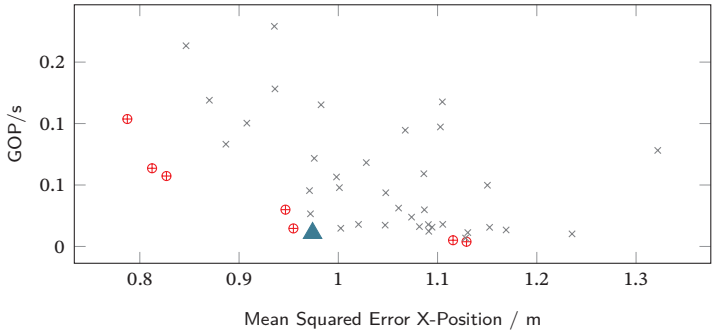


Abbildung 6.10: Ergebnisse der Hyperparametersuche für RoNIN-Datensatz

2, Dilationen (1, 8), drei Stapeln und einer Dropout-Rate von 0,1 dar. Diese bietet einen MSE von 0,97 m (X-Position) bzw. 1,25 m (Y-Position) sowie eine Rechenkomplexität von 10,8 Millionen Operationen. Basierend auf den Evaluationsergebnissen von LOTTA, kann der Beschleuniger die Inferenz eines Stapels von 400 Zeitschritten demzufolge in etwa 90,1 ms durchführen. Daraus folgt, dass LOTTA Eingangsdaten mit einer Frequenz von 4,439 kHz verarbeitet und folglich im Zusammenspiel mit dem Low-Power-FPGA gut für den Einsatz in diesem Anwendungsfall geeignet ist.

Sprachsteuerung

Eine häufige Anwendung für KI im IoT ist die Erkennung von Schlüsselwörtern in Audiosignalen zur Sprachsteuerung. Um den Einsatz von DNNs in solchen Geräten zu ermöglichen, sind in diesem Fall ebenfalls kleine Netzwerke und energieeffiziente Hardware erforderlich. Im Gegensatz zur Trägheitsnavigation erfordert die Sprachsteuerung lediglich die Verarbeitung der Daten eines Sensors, dem Mikrofon. Außerdem ist hier keine direkte Verarbeitung der Daten erforderlich, sodass die Anforderungen an den minimalen Datendurchsatz etwas geringer sind. Dennoch sollte eine gewisse Latenz erzielt werden, um eine zeitnahe Reaktion auf die Sprachbefehle zu zeigen. Somit stellt dieser Anwendungsfall andere Anforderungen an das System und damit auch an die Architektur des Beschleunigers verglichen mit der Trägheitsnavigation.

In diesem Fall dient der SpeechCommands-Datensatz [234] als Basis für das Training und die Evaluation der TCN-Hyperparameter. Abbildung 6.11 zeigt

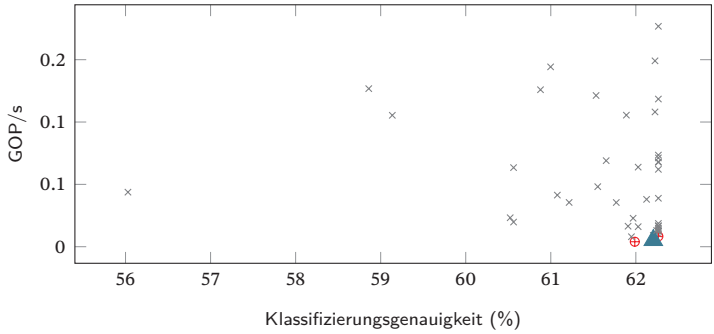


Abbildung 6.11: Ergebnisse der Hyperparametersuche für SpeechCommands-Datensatz

die Ergebnisse der Exploration hinsichtlich der Klassifizierungsgenauigkeit und der Rechenkomplexität für jeden der 60 untersuchten Hyperparameterkombinationen. Als guten Kompromiss ergibt sich dabei die Konfiguration mit 24 Filtern, Kernelgröße von eins, Dilationen (1, 2), drei Stapeln und einer Dropout-Rate von 0,2. Diese Konfiguration bietet eine Klassifizierungsgenauigkeit von 62,21 % und eine Rechenkomplexität von 5,2 Millionen Operationen. Im Ergebnis kann LOTTA die Inferenz eines Stapels von 400 Zeitschritten in etwa 43,5 ms durchführen. Daraus resultiert eine Frequenz von 9,195 kHz mit der die Verarbeitung von Eingangsdaten realisiert werden kann. Diese liegt damit unterhalb der gewöhnlichen Abtastrate von 44,1 kHz aufgrund des Hörbereichs von 0,02 kHz bis 20 kHz. Allerdings ist eine Abdeckung des gesamten Spektrums für die Sprachsteuerung nicht notwendig, da der Tonumfang menschlicher Stimmen deutlich beschränkter ist. Hirsch et al. [235] haben nachgewiesen, dass Systeme bereits mit einer Abtastfrequenz von 8 kHz gesprochene Wörter mit geringer Fehlerrate erkennen.

Zusammenfassend belegen die beiden Fallstudien die Eignung von LOTTA für den Einsatz in unterschiedlichen Anwendungsfällen im IoT. Die Implementierung von Low-Power-FPGAs in solchen Geräten mit Anforderungen an eine hohe Energieeffizienz und eine geringe Leistungsaufnahme ist somit nachweislich möglich und bietet zeitgleich die Möglichkeit von Architekturanpassungen zur Laufzeit im Falle neuartiger KI-Ansätze.

Kapitel 7

Technologische Optimierung für sicherheitskritische Anwendungen

Die zuvor betrachteten Ansätze im Hardware/Software Co-Design auf System- und RTL-Ebene ermöglichen ein optimiertes Mapping der Anwendungen auf das System hinsichtlich Performance, Energieeffizienz und Linkbandbreite. Wie bereits in Kapitel 4 erwähnt, können darüber hinaus weitere Maßnahmen durchgeführt werden, um zusätzliche nicht-funktionale Anforderungen an das System möglichst effizient zu implementieren. Insbesondere der Umgang mit transienten Fehlern in sicherheitskritischen Fehlern stellt eine Herausforderung auf verschiedenen Ebenen dar. Wie beispielsweise in den Publikationen [Kem22, Hoe23a] vorgeschlagen, können Optimierungen auf Anwendungs- und Systemebene durchgeführt werden, um Fehler abzufangen bzw. deren Einfluss zu verringern.

Allerdings funktionieren diese Techniken nicht bei einem unerwarteten Spannungsabfall im System. Um in solchen Fällen einen Neustart des Gesamtsystems zu vermeiden, bieten sich Anpassungen auf Technologieebene an. Im Folgenden wird daher ein Konzept basierend auf nichtflüchtigen Speicherzellen präsentiert, welches eine Lösungsmöglichkeit für dieses Problem darstellt. Die Erläuterungen und Ergebnisse wurden vom Autor in [Kre23d] veröffentlicht.

7.1 Motivation

Eingebettete Systeme in sicherheitskritischen Bereichen unterliegen hohen Anforderungen in Bezug auf Zuverlässigkeit und Leistung. Daher stützen sich gängige Rechenplattformen auf räumliche Redundanz, um Fehler zu erkennen und je nach Implementierung nach Möglichkeit zu beheben. Dies

geht jedoch auf Kosten eines großen Flächenbedarfs und einer geringeren Systemleistung. Aufgrund des Aufkommens gemischt-kritischer Systeme, bei denen sicherheitskritische und nicht-kritische Anwendungen auf der gleichen Hardwareplattform verarbeitet werden, bedarf es Hardwarekonzepten, die trotz Implementierung von Sicherheitsmechanismen eine hohe Performance bieten.

Eine solche Möglichkeit bieten unter anderem Lockstep-Prozessoren, welche von Gizopoulos et al. [236] vorgestellt wurden. Dabei wird eine sicherheitskritische Aufgabe redundant auf zwei oder mehr Prozessoren ausgeführt und das Ergebnis aller Kerne verglichen, um Fehler zu erkennen. Kommt es zu abweichenden Ergebnissen zwischen den Kernen, erfolgt entweder eine Wiederholung der Berechnungen oder es wird der Wert als finales Ergebnis der Operation ausgewählt, welcher mehrfach vorkommt (Majority Vote). Nicht-kritische Anwendungen werden dagegen nicht redundant ausgeführt, wodurch sich eine höhere Systemperformance erzielen lässt. Allerdings erfordert dieser Ansatz eine enge Kopplung der Prozessoren auf Hardwareebene, worunter die Flexibilität des Systems leidet.

Um eine größere Flexibilität zu erreichen, können auf Checkpointing basierende Konzepte zurückgegriffen werden [237, 238], die die Ausführung von Aufgaben auf räumlich getrennten Kernen ermöglichen. Dennoch decken diese Gegenmaßnahmen in der Regel keine Stromausfälle ab, da die Architekturen ausschließlich auf flüchtigen FFs basieren.

In der Zwischenzeit wurden mehrere neuartige nichtflüchtige Speichertechnologien entwickelt, die kurze Lese- und Schreibzeiten bieten und daher schnelle und aggressivere Power-Gating-Strategien ermöglichen. Infolgedessen ergaben sich innerhalb des vergangenen Jahrzehnts einige neue Ansätze aus der Forschung, die sich diese Technologien zu Nutze gemacht haben, um nichtflüchtige Register oder Prozessoren zu entwickeln. Darüber hinaus bieten sich damit neue Möglichkeiten für Anwendungen mit extrem niedrigem Stromverbrauch an. So wurde in diesem Rahmen unter anderem auch ein neues Paradigma, das Normally-Off Computing [239], abgeleitet. Dieses verfolgt das Ziel, das eingebettete System möglichst lange im ausgeschalteten Zustand zu halten, um während der gesamten Lebensdauer des Systems Energie zu sparen. Es wird nur folglich dann von außen aktiviert, wenn eine Berechnung gewünscht wird. Aufgrund der hohen Lese- und Schreibgeschwindigkeit der neuartigen nichtflüchtigen Speichertechnologien kann der aktuelle Systemzustand ohne lange Wartezeit vor dem Ausschalten abgespeichert und nach dem Einschalten geladen werden. Reine nichtflüchtige Logik-Implementierungen weisen allerdings einen höheren Energiebedarf

beim Lesen und Schreiben sowie eine geringere Lebensdauer auf. Daher ist die reine Verwendung dieser Speicherzellen in eingebetteten Hochleistungsplattformen wenig vorteilhaft.

Die Kombination von Checkpointing und nichtflüchtige Register in Form von hybriden flüchtigen/nichtflüchtigen Registern bietet sich folglich an, um eine schnelle Systemwiederherstellung im Falle eines Stromausfalls zu ermöglichen und gleichzeitig die Leistung hoch zu halten [240]. Darüber hinaus kann dieser Ansatz auch für schnelle Kontextwechsel zwischen nicht-kritischen Anwendungen ausgenutzt werden, da die Speicherzellen im Betrieb die Erstellung von Checkpoints und die Durchführung eines Rollbacks in kurzer Zeit erlaubt [241]. Dennoch ist das Ersetzen aller CMOS-Register in einer digitalen Schaltung in vielen Anwendungen keine praktikable Lösung, da die hybriden Register einen deutlich höheren Flächenverbrauch aufweisen [242]. Insbesondere CMOS-Register eingebetteter Systeme, in denen die verfügbare Fläche auf dem SoC in der Regel stark begrenzt ist, können folglich nicht vollständig ersetzt werden. Es werden daher Ansätze benötigt, um eine minimale Menge an CMOS-Registern zu ersetzen bei Erhalt der Möglichkeit von Checkpointing und Rollback.

Nachfolgend wird das Konzept einer Toolchain präsentiert, welche automatisch nach der Menge an FFs in einer Netzliste sucht, um diese Funktionalität für sicherheitskritische Anwendungen in die bestehende Logik zu integrieren. Dafür synthetisiert die Toolchain zunächst den RTL-Code einer gegebenen Anwendung und analysiert die daraus resultierende Netzliste hinsichtlich aller potenziell zustandserhaltenden FFs im Design. Anschließend werden diese durch hybride flüchtige/nichtflüchtige FFs ersetzt und an die Checkpointing-Infrastruktur des Systems verbunden, um die Erstellung eines Checkpoints oder die Durchführung eines Rollbacks auszulösen. Daraus ergibt sich als Ausgabe der Toolchain eine Netzliste, welche eine optimierte Kombination von Checkpointing und hybriden flüchtigen/nichtflüchtigen FFs für den Einsatz in sicherheitskritischen Anwendungen bietet.

7.2 Hintergrund

7.2.1 Magnetischer Tunnelkontakt

Wie zuvor erwähnt, haben Forscher innerhalb der letzten Jahrzehnte enorme Entwicklungssprünge im Bereich neuartiger nichtflüchtiger Speichertechnologien erzielt. Die verschiedenen Technologien, die daraus hervorgegangen

sind, wie beispielsweise MRAM, RRAM oder PCM, unterschieden sich zum Teil stark in ihren Charakteristika [243, 244]. Für den Einsatz dieser Technologien auf FF- bzw. Registerebene sind einerseits der Flächenverbrauch der Zelle und andererseits die Lese- und Schreibzeiten von hoher Bedeutung. Während die Größe der Zelle entscheidende Auswirkung auf die Realisierbarkeit des SoCs hat, bestimmen die Zugriffszeiten die Performance hinsichtlich der Erstellung eines Checkpoints sowie der Durchführung eines Rollbacks. Gleichzeitig muss die Speicherzelle für eine hohe Anzahl an Schreibzugängen ausgelegt sein, um Checkpointing in einer möglichst hohen Frequenz durchzuführen.

Folglich eignet sich die MRAM-Technologie am besten für die Entwicklung hybrider FFs, im Folgenden als hybride Magnetischer Flip-Flops (MFFs) bezeichnet, da diese im Gegensatz zu RRAM-Zellen eine höhere Performance und Lebensdauer bei nur minimal größerem Flächenverbrauch bietet [245]. Der Magnetische Tunnel-Kontakt (MTK), auf dem diese Technologie basiert [246], nutzt einen magnetoresistiven Effekt, den TMR-Effekt, um Informationen zu speichern. Solche Komponenten bestehen aus zwei ferromagnetischen Schichten, welche durch eine dielektrische Tunnelbarriere voneinander getrennt sind. Die isolierende Schicht ist dabei nur wenige Nanometer dünn, um das Tunneln von Elektronen durch diese zu ermöglichen. Eine der ferromagnetischen Schichten hat eine konstante magnetische Polarisation und wird als Referenzschicht (Reference Layer) verwendet, während die Polarisation der anderen Schicht verändert werden kann (Free Layer). Wenn beide Schichten die gleiche Polarisation aufweisen, befinden sie sich im parallelen Zustand, wodurch ein relativ geringer magnetischer Tunnelwiderstand entsteht. Grund dafür ist, dass die Elektronen mit Majoritätsspin und Minoritätsspin leicht zur zweiten ferromagnetischen Schicht tunneln und identische Majoritäts- bzw. Minoritätszustände einnehmen. Im Gegensatz dazu ist dies im antiparallelen Zustand nur mit größerem Energieaufwand möglich. Dabei füllen die Elektronen mit Majoritätsspin und Minoritätsspin aus der ersten ferromagnetischen Schicht die Minoritäts- bzw. Majoritätszustände in der zweiten ferromagnetischen Schicht, was zu einem höheren Tunnelwiderstand führt. Wie in Abbildung 7.1 dargestellt, ergibt sich so für den magnetischen Tunnelwiderstand eine temperaturabhängige R-V-Hysteresekurve, welche einen Zustand mit hohem Widerstand und einem mit geringerem Widerstand umfasst. Durch die binäre Kodierung dieser beiden Zustände ist folglich eine direkte Anwendung in digitalen Systemen möglich.

Die Implementierung solcher Zellen auf einem Chip kann auf unterschiedliche Arten erfolgen, wobei sich diese in der physikalischen Realisierung des Mechanismus zur Veränderung der Orientierung des magnetischen Felds im

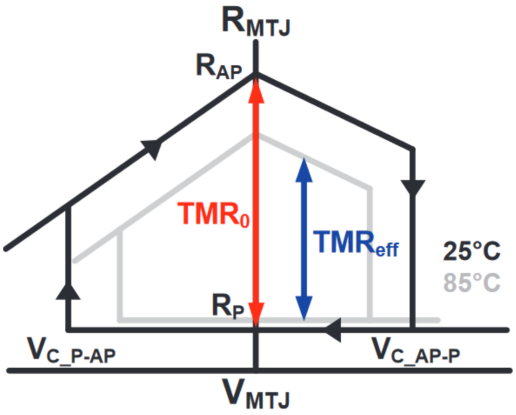


Abbildung 7.1: R-V-Hysteresekurve des magnetischen Tunnelwiderstands aus [247]

Free Layer unterschieden. Dadurch ergeben sich folglich Unterschiede insbesondere in Performance, Energiebedarf und Lebensdauer der Zellen [110]. Relevant für den Einsatz in SoCs bestehend aus verschiedenen Komponenten ist hierbei letztlich die Kompatibilität zu bestehenden Fertigungsprozessen. Unter anderem haben Rossi et al. [248] nachgewiesen, dass MTK-Zellen kompatibel zu CMOS und damit geeignet für den Einsatz in gewöhnlichen SoC Architekturen sind.

7.2.2 Register Checkpointing

Das grundlegende Prinzip des Checkpointing besteht darin, den aktuellen Zustand des Systems nach oder während der Ausführung von Aufgaben zu speichern. Tritt ein Systemfehler auf, kann ein Rollback durchgeführt werden, um zu einem sicheren Zustand zurückzukehren und die seit dem letzten Checkpoint bearbeiteten Aufgaben zu wiederholen. Dieser Ansatz bietet den Vorteil, dass ein vollständiges Zurücksetzen des Systems nicht notwendig ist, d.h. das System muss keinen Neustart durchführen. Auf diese Weise lässt sich insbesondere in sicherheitskritischen Systemen die Zeit verkürzen, die das System benötigt, um sich wiederherzustellen und dort fortzufahren, wo der Fehler aufgetreten ist.

Allgemein kann Checkpointing in Hardware sowie in Software implementiert werden. Bei letztgenanntem konzentriert sich die Forschung im Wesentlichen

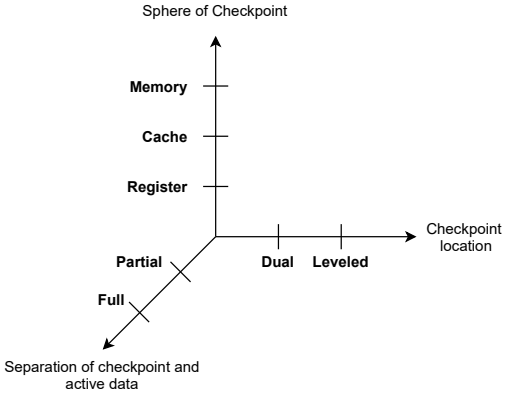


Abbildung 7.2: Checkpoint Taxonomie nach Teodorescu et al. aus [Kem22]

auf die Bestimmung eines günstigen Zeitpunkts zur Erstellung eines Checkpoints hinsichtlich des Einflusses auf die Performance des Systems [249] oder der Energieeffizienz [250]. Dieser Ansatz bietet zwar eine hohe Flexibilität im Gegensatz zu reinen Hardwareimplementierungen, birgt aber auch Nachteile. Während des Kopierens der Daten aus dem Registersatz des Prozessors in einen bestimmten Speicher können meist keine Berechnungen auf dieser Einheit ausgeführt werden. Zudem sorgen die Datentransfers zu einer erhöhten Auslastung des Bussystems.

Die Implementierung von Checkpointing in Hardware lässt sich hinsichtlich der Performance und Energieeffizienz wesentlich effizienter realisieren als in Software. Eine generelle Klassifikation von Checkpointing und Rollback in Hardware kann anhand der Taxonomie von Teodorescu et al. [238] vorgenommen werden. Diese ist grafisch in Abbildung 7.2 dargestellt. Dabei definiert die y-Achse die in das Checkpointing einbezogenen Speicherebenen, wobei diese Sphäre die gesamte Hierarchie bis zum höchsten Speicherelement einschließt. Checkpointing auf Cache-Ebene verlangt folglich auch die Sicherung der Informationen auf Registerebene. Wohin diese Checkpoints gespeichert werden, beschreibt die x-Achse der Taxonomie. In diesem Fall wird zwischen der Speicherung in einer niedrigeren Ebene der Speicherhierarchie (Leveled) und der Sicherung auf derselben Ebene durch die Verwendung zusätzlicher Hardwarekomponenten (Dual) unterschieden. Als letzten Aspekt beschreibt die Taxonomie die Trennung des Checkpoints von den aktiv verwendeten Daten, welche entweder vollständig (Full) oder in einem partiellen Schema (Partial) erfolgt.

In Hardware sorgen folglich zusätzliche Komponenten für die Realisierung von Checkpointing und Rollback. Allerdings führen diese Ansätze zwangsläufig zu einem erhöhten Flächenbedarf, weshalb dessen Reduzierung ein wesentliches Optimierungsziel entsprechender Hardwarekonzepte ist. So schlagen beispielsweise Li et al. [251] ein feinkörniges Register Checkpointing vor, das den Hardware-Overhead reduziert, indem es nicht den Zustand des gesamten Registersatzes, sondern nur den einiger weniger Register sichert. Dafür wird der Wert im Register bei einem Schreibvorgang zusammen mit dessen Adresse in den Checkpoint-Speicher geschrieben, welcher als Stack ausgelegt ist. Folglich enthält dieser Speicher nur die veränderten Werte seit dem letzten Checkpoint, welche im Falle eines Rollbacks direkt in den Registersatz zurückgeschrieben werden. Die Dimensionierung des Stacks erfolgt zur Design-Zeit durch Profiling der erwarteten Anwendungen, was ein klarer Nachteil dieses Konzepts darstellt. Sobald sich die Anwendung ändert, ist der Speicher möglicherweise nicht mehr in der Lage, alle erforderlichen Register der Registerdatei zu speichern. Somit ist dieser Ansatz nur für den Einsatz in Systemen geeignet, deren Anforderungen sich zur Laufzeit nicht verändern.

Eine in dieser Hinsicht flexiblere Methodik schlagen Kempf et al. [Kem22] vor. Dabei werden Checkpoints erstellt, sobald es zu einem Cache Miss kommt und ein Überschreiben aktiver Daten zwangsweise erfolgt. Wie zuvor dargelegt, umfasst die Erstellung eines Checkpoints sämtliche darunterliegenden Speicherebenen, weshalb auch der Registersatz im Prozessor zu berücksichtigen ist. Die vorgeschlagene Architektur basiert auf einem redundanten Registersatz, der von zwei Registern verwaltet wird. Dies dient der Bestimmung der Speicherbank des zu lesenden Registers sowie der Anzeige von Datenänderungen seit dem letzten Checkpoint. Die Durchführung von Checkpointing hat keinen Einfluss auf die Performance des Systems, da die Erstellung des Checkpoints nahtlos im laufenden Betrieb erfolgt. Allerdings ist zu berücksichtigen, dass dieser Ansatz einen höheren Flächenbedarf erfordert als das zuvor vorgestellte Konzept.

7.2.3 Verwandte Arbeiten

Abgesehen von den genannten Ansätzen zur Implementierung von Checkpointing auf Registerebene wurden mehrere Ansätze veröffentlicht, die auf nichtflüchtigen Registern basieren. Bedingt durch die begrenzte Anzahl an Schreibzugriffen und der geringeren Performance im Vergleich zu CMOS, werden häufig nichtflüchtige Register in den SoC hinzugefügt ohne die flüchtigen

Register zu ersetzen. Um die Anzahl der für das Checkpointing erforderlichen nichtflüchtigen Register und damit den Platzbedarf des Systems zu verringern, verwenden einige der vorgeschlagenen Architekturen einen Codec zur Komprimierung der Daten [252, 253]. Dies führt jedoch zu einem beträchtlichen Zeitaufwand für die Erstellung von Checkpoints und das Rollback, was den Anforderungen von sicherheitskritischen Anwendungen wie dem autonomen Fahren nicht in jedem Fall entspricht.

Darüber hinaus haben sich bislang nur wenige Forschungsgruppen mit der Kombination von CMOS FFs und MTK in eine einzelne logische Komponente beschäftigt. Beispielsweise haben Senni et al. [254] einen solchen nichtflüchtigen FF basierend auf dem MTK für schnelles Checkpointing vorgestellt. Sie zeigen, dass die Energieeffizienz insbesondere für Normally-Off Computing verbessert wird, beachten in ihrer Architektur allerdings nicht die begrenzte Lebensdauer von MTKs. Darüber hinaus basiert ihr Ansatz auf dem vollständigen Ersetzen aller Register durch nichtflüchtige FFs, was zu einem signifikanten Anstieg des Flächenverbrauchs führt.

Um dieses Problem anzugehen, entwickelten Gebregiorgis et al. [255] einen Ansatz, um eine kleinere Anzahl von Registern zu ersetzen und gleichzeitig die Checkpointing-Funktionalität beizubehalten. Die Identifikation der zustandserhaltenden FFs basiert dabei auf einer Gate-Level Simulation. Dabei injiziert und verfolgt ein Analysemodul gezielt Fehler in FFs, um abzuschätzen, ob das entsprechende FF für einen Rollback gesichert werden muss. Insgesamt wird damit eine deutliche Verringerung des Flächenverbrauchs erzielt, allerdings auf Kosten einer hohen Simulationszeit für die Analyse.

7.3 Hybrider Magnetischer Flip-Flop

Die zuvor vorgestellten Forschungsergebnisse belegen, dass der Einsatz hybrider MFFs für die Realisierung von Checkpointing in Hardware sinnvoll ist. Sie sorgen nicht nur dafür, dass persistente Checkpoints erstellt werden, sondern ermöglichen auch die Anwendung aggressiver Power-Gating Strategien für das Normally-Off Computing. Im Rahmen der vorliegenden Arbeit wurden weitere Optimierungen dieser Ansätze untersucht, wobei das Register-Checkpointing durch die Kombination von CMOS-Technologie und MTKs erzielt wird.

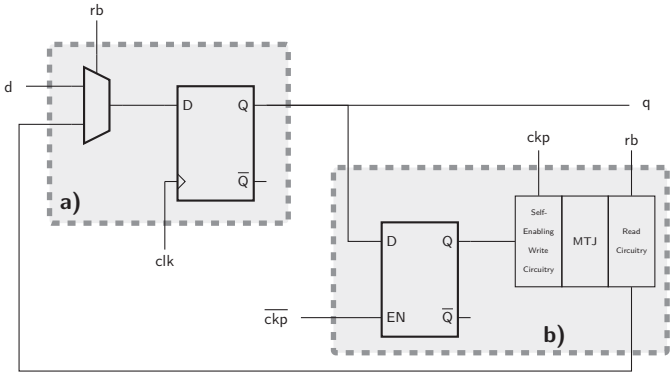


Abbildung 7.3: Strukturelle Übersicht des hybriden MFF

Die vorgeschlagene High-Level Architektur des hybriden MFFs ist in Abbildung 7.3 dargestellt. Neben den beiden Datensignalen *d* und *q*, verfügt die Zelle über zwei Kontrollsignale. Während *ckp* die Erstellung eines Checkpoints triggert, wird das Signal *rb* genutzt, um einen Rollback durchzuführen. Um zu vermeiden, dass identische Logik in mehrfacher Ausführung in der Schaltung vorhanden ist, erfolgt die Ansteuerung dieser Signale von außerhalb. Dies erlaubt dabei gleichzeitig die Implementierung komplexerer Checkpointing-Strategien, da dieser Ansatz sehr gut skalierbar ist.

Das Design des hybriden MFFs ist abgeleitet von der Architektur, die Chabi et al. [256] vorgeschlagen haben. Ihr Ansatz unterstützt dabei auch die Multi-Bit Integration in die Zelle, wodurch sich der Flächenverbrauch durch das Einfügen der MFFs signifikant verringert. Dieser wird nachfolgend allerdings nicht betrachtet. Wie bereits erwähnt, ist einer der begrenzenden Faktoren beim Einsatz von MTKs die limitierte Anzahl an möglichen Schreiboperationen im Vergleich zu SRAM. Aus diesem Grund verfügt die Zelle über einen sogenannten Self-Enable Schaltkreis, der nur dann eine Schreiboperation initiiert, wenn der zu schreibende Zustand vom aktuellen Zustand der Zelle abweicht. Insgesamt ist das von Chabi et al. vorgeschlagene Design der Zelle allerdings nicht für hohe Performance optimiert, da der Prozessor angehalten werden muss, um einen Checkpoint zu erstellen.

Aus diesem Grund verfügt die in Abbildung 7.3 dargestellte Architektur des MFFs über einen zusätzlichen D-Latch (s. Abbildung 7.3 (b)), welcher den aktuellen Zustand des CMOS FF zwischenspeichert. Dies ist notwendig, da die Lese- und Schreibzeiten für den MTK deutlich höher sind als für einen CMOS

FF. Folglich müssen die Signale für Checkpointing oder Rollback deutlich länger aktiv sein als eine Taktperiode. Dementsprechend erlaubt das hier vorgeschlagene Design der Zelle eine unterbrechungsfreie Erstellung von Checkpoints, was insbesondere für die Anwendung von hybriden MFFs in hochperformanten Systemen von Relevanz ist.

7.4 Toolchain

In der zuvor erwähnten Arbeit von Chabi et al. [256] geben die Autoren für ihre Architektur eines hybriden MFFs einen Flächenverbrauch von 300 % im Vergleich zu konventionellen CMOS FFs an. Folglich ist das Ersetzen aller Zellen hinsichtlich der benötigten Fläche nicht sinnvoll. Stattdessen sollten lediglich zustandserhaltende FFs durch hybrides MFFs für die Implementierung von Checkpointing und Rollback ersetzt werden, um den Overhead einzugrenzen. Zustandserhaltende FFs sind wie folgt definiert:

Definition 8 (Zustandserhaltendes Flip-Flop)**.** *Ein zustandserhaltendes FF ist ein Element in der Menge von Registern in einem digitalen System, dessen interner Zustand bei der Erstellung eines Checkpoints zwingend gesichert werden muss, um einen Rollback zu ermöglichen.*

Im Gegensatz dazu müssen Pipeline-Register und Pufferspeicher nicht gesichert werden, wie von Darbari et al. nachgewiesen [257], da sich deren Daten von dem Zustand anderer Register ableiten und somit fehlerfrei wiederherstellen lassen. Aus diesem Grund wird nachfolgend eine Methodik basierend auf frei verfügbaren Tools vorgestellt, welche automatisiert alle potenziell zustandserhaltenden FFs in einer gegebenen Netzliste sucht und diese durch hybride MFFs ersetzt. Ein Überblick dieser Toolchain ist in Abbildung 7.4 gegeben.

Hierbei muss allerdings zunächst erwähnt werden, dass die vorgestellte Methodik nicht garantiert, nur zustandserhaltende FFs in einer Netzliste zu finden. Es wird in der Regel ein paar wenige FFs geben, welche für den Rollback irrelevant sind. Grund dafür ist, dass sich die exakte Rekonstruktion einer FSM aus einer Netzliste ohne weitere anwendungsspezifische Informationen schwierig gestaltet. So existiert im aktuellen Stand der Technik bislang keine Methodik, welche für diesen Fall alle und ausschließlich zustandserhaltende FFs in einer digitalen Schaltung identifiziert [258]. Dennoch zeigen die Evaluationsergebnisse dieser Arbeit, dass der Ansatz alle relevanten FFs in

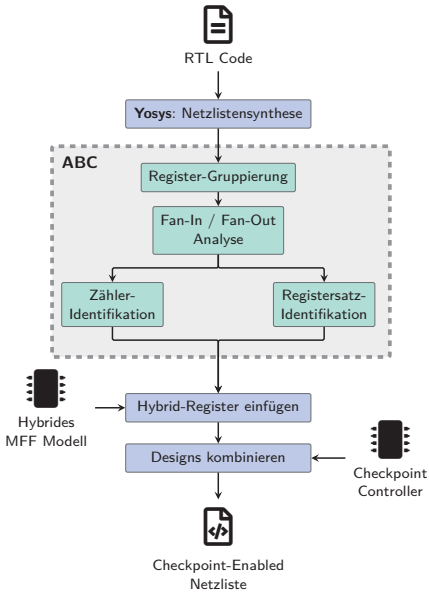


Abbildung 7.4: Überblick über die Toolchain zum Erkennen und Ersetzen von zustandserhaltenden FFs

RISC-V basierten Prozessorkernen findet und somit ein Rollback in solchen Systemen ermöglicht.

Die Algorithmen zur Erkennung von zustandserhaltenden FFs sind in ABC [259] implementiert, einem in der akademischen Welt weit verbreiteten Tool zur Optimierung und Konvertierung von Logik. Bevor ABC ein digitales Hardwaredesign verarbeiten kann, muss dieses zunächst in eine BLIF-basierte Netzliste synthetisiert werden. Hierfür kommt die Yosys Verilog-Synthese [260] zum Einsatz, welche aus dem gegebenen RTL-Code eine entsprechende Netzliste generiert. Diese wird anschließend in ABC geladen, wobei das Tool automatisiert einen Graph bestehend aus den FFs in der Schaltung und einen And-Inverter-Graph für die kombinatorische Logik erstellt. Letzterer ist für die Erkennung der zustandserhaltenden FFs irrelevant und muss daher nicht beachtet werden.

Um die in Abbildung 7.4 dargestellten vier Schritte der Erkennung von zustandserhaltenden Registern zu implementieren, benötigt ABC Informationen über die Fan-Ins und Fan-Outs der vorhandenen FFs. Dafür muss als

vorbereitende Maßnahme jedes einzelne FF zu Registern gruppiert werden. Anhand dieser Informationen lassen sich anschließend die Fan-In- und Fan-Out-Beziehungen zwischen den Registern sowie die interne Verdrahtung der FFs untereinander ableiten. Basierend auf den Relationen erfolgt im nächsten Schritt die Suche nach Zählern und Registersätzen in der gegebenen Netzliste. Ebenjene Komponenten sind für die Implementierung einer FSM erforderlich und bilden deren Zustand ab. Dabei stellen Zähler häufig einen Teil des Kontrollflusses dar, wie dies beispielsweise in Prozessorkernen der Fall ist. Hier gibt der Wert des Befehlszählers vor, welche Instruktion im nächsten Takt in die Prozessorpipeline geladen werden soll. Folglich ist die Kenntnis über den Zählerstand für das korrekte Zurücksetzen der Schaltung unabdingbar, weshalb ein Checkpoint zwingend den Zustand des Befehlszählers enthalten muss. Auch die Daten des Registersatzes werden für einen erfolgreichen Rollback benötigt, da dieser Zwischenergebnisse und geladene Operanden für die nachfolgenden Instruktionen enthält.

Nachdem alle Register verarbeitet wurden, gibt die Toolchain eine Liste aller potenziell zustandserhaltenden FFs aus. Diese Liste wird bei der weiteren Verarbeitung außerhalb von ABC verwendet, um die hybriden MFFs in die Netzliste einzufügen. Für das Durchführen von Checkpointing und Rollbacks wird darüber hinaus ein passendes Steuermodul benötigt, der Checkpoint Controller, welcher die hybriden MFF direkt ansteuert. Aus diesem Grund verfügen die hybriden MFFs über zwei zusätzliche Ports im Vergleich zu gewöhnlichen CMOS FFs. Im finalen Schritt wird der entworfene Checkpoint Controller mittels Yosys synthetisiert und die daraus resultierende Netzliste in das bestehende Design integriert. Folglich erhält der Benutzer als Ausgabe eine Netzliste, welche Checkpointing basierend auf hybriden MFFs auf Registerebene ermöglicht.

7.4.1 Register-Gruppierung

Während der Synthese einer Anwendung wird eine Netzliste erstellt, die nur generische Grundkomponenten wie FFs oder Gatter verwendet, um das definierte Verhalten in Hardware zu implementieren. Infolgedessen werden auch die vorhandenen Register in mehrere FFs aufgeteilt. Um diesen Schritt für die weiteren Analysen rückgängig zu machen, muss zunächst eine Gruppierung der FFs zu Registern erfolgen. Da jedoch jedes dieser FFs während der Synthese mit einem Namen versehen wird, wird die Rekonstruktion von Registern direkt durch Iteration der Komponenten in der Netzliste und Analyse ihres Na-

mens durchgeführt. Dies geschieht in der ersten Phase der vorgeschlagenen Toolchain.

7.4.2 Fan-In- und Fan-Out-Analyse

Neben dem Namen, der den FFs in der Netzliste gegeben wird, bedarf es zusätzlich der Analyse der Fan-Ins und Fan-Outs für jede darin enthaltenen Komponente. Hierdurch lassen sich die Verknüpfungen zwischen einzelnen FFs ohne Betrachtung der kombinatorischen Logik zwischen diesen Elementen in der digitalen Schaltung erkennen. Fan-Ins beschreiben dabei Vorgängerknoten eines FFs im Graph und Fan-Outs nachfolgende FFs. Diese Information ist nicht nur für die Implementierung des Designs auf Hardware von Interesse, sondern wird auch nachfolgend für die Identifizierung von Zählern und Registersätzen benötigt. Dabei ist es nicht nur notwendig, die Fan-Ins und Fan-Outs zwischen den Registern, sondern auch innerhalb eines Registers selbst zu analysieren.

7.4.3 Zähler-Identifikation

Für die Erkennung von Zählern in der Netzliste sind die registerinternen Relationen zwischen den FFs relevant. Folglich wird in diesem Fall durch die gefundenen Register iteriert und diese so nacheinander geprüft. Ein Zähler lässt sich, wie von Subramanyan et al. [261] nachgewiesen, daran erkennen, dass der Wert eines Bits B_i nur von den niederwertigeren Bits und dem eigenen Zustand abhängt. Der mathematische Zusammenhang lässt sich, wie in Gleichung (7.1) dargestellt, formulieren.

$$B_i = f(B_i, B_{i-1}, ..., B_1, B_0) \tag{7.1}$$

Folglich sucht der Algorithmus in der Netzliste nach Registern, welche diese Anforderung hinsichtlich ihrer Fan-Outs erfüllen. Sobald ein solcher Zähler gefunden wurde, werden die zugehörigen FFs als zustandserhaltend markiert und somit am Ende der Toolchain entsprechend ersetzt.

7.4.4 Registersatz-Identifikation

Im Gegensatz zu Zählern gibt es innerhalb des Registersatzes keine internen Abhängigkeiten zwischen den Registern. Der Inhalt ändert sich nur auf der Grundlage von eingehenden Daten von außerhalb des Blocks. Daher werden Register mit internen Beziehungen in diesem Algorithmus bereits von vornherein für diese Suche ausgeschlossen.

Normalerweise basiert die Lese- und Schreiblogik von Registersätzen auf einer Baumstruktur, die nur aus Multiplexern zur Dekodierung der Zugriffsadresse besteht. Zusätzliche Logik, welche die Daten selbst modifiziert, ist in der Regel nicht vorhanden. Um einen Registersatz zu identifizieren, muss der Algorithmus folglich nach einer solchen Struktur innerhalb der Netzliste suchen. Dies erfolgt durch die Analyse der Fan-Ins und Fan-Outs der einzelnen Register. Dabei lassen sich alle Register, welche exakt dieselben Fan-Ins und Fan-Outs haben, zu einem Registersatz zusammenfassen. Ähnlich der Zähleridentifikation werden schließlich alle im Registersatz enthaltenen FFs als zustandserhaltend markiert.

7.5 Evaluation

Um zu demonstrieren, dass die vorgeschlagene Toolchain korrekt funktioniert, werden leicht nachvollziehbare Hardwarearchitekturen benötigt. Das bedeutet, dass bekannt sein muss, welche FFs zustandserhaltend sind und welche von der Toolchain nicht als solche markiert werden sollten. Folglich erlaubt dieses Vorgehen eine manuelle Verifikation der Ergebnisse.

Zum einen wird der in C entworfene Comet Prozessor [262] untersucht, welcher über einen 32-bit RISC-V Kern verfügt. Er besteht aus einer fünfstufigen Pipeline mit Forwarding und der Unterstützung für Multi-Cycle Operationen. Da Yosys Verilog Code als Eingabe für die Synthese erwartet, kam für die Generierung der Hardwarebeschreibung aus dem C Code Catapult HLS von Siemens EDA [263] zum Einsatz.

Darüber hinaus untersucht diese Arbeit die Toolchain für zwei weitere RISC-V Kerne, welche in Verilog verfügbar sind: den PicoRV32 [264] sowie den tinyriscv [265]. Erstgenannter Prozessor wurde bei seiner Optimierung hinsichtlich hoher Taktfrequenz und geringem Flächenbedarf optimiert und verfügt daher lediglich über eine minimale Anzahl an Hardwaremodulen. Der tinyriscv

Prozessor basiert auf einer dreistufigen Pipeline und besitzt zusätzlich im Vergleich zum PicoRV32 einen dedizierten Divider sowie Status- und Steuerregister.

7.5.1 Verifikation

Zur Sicherstellung der korrekten Funktionstüchtigkeit der vorgeschlagenen Toolchain müssen zwei Schichten betrachtet werden: Technologie und System. Auf technologischer Ebene ist zu verifizieren, dass die vorgeschlagene Architektur des hybriden MFFs die Daten korrekt verarbeitet. Hierfür wurde auf eine SPICE Simulation unter Verwendung eines MTK-Modells basierend auf der Arbeit von Kim et al. [247] zurückgegriffen. Die Experimente konnten dabei die Korrektheit des hybriden MFFs nachweisen [Zha22a].

Auf Systemebene ist dagegen eine Verhaltenssimulation ausreichend, um das Design zu verifizieren. Aus diesem Grund wurde hierfür die Architektur des hybriden MFF mittels Verilog beschrieben und daraus eine Netzliste synthetisiert. Zusätzlich erfordern die Untersuchungen einen Checkpointing-Controller als Komponente der Testbench, welcher Checkpointing- und Rollback-Signale periodisch generiert. Der Einfachheit halber basiert dieser hier auf einem einfachen Zähler. Die durchgeführten Experimente demonstrieren, dass die Architektur des hybriden MFFs einen korrekten Rollback basierend auf dem letzten Checkpoint ermöglicht [He22b].

7.5.2 Ergebnisse

Im Folgenden werden die Ergebnisse der Toolchain für die zuvor genannten Prozessoren präsentiert und diskutiert. Eine Übersicht der Resultate ist in Tabelle 7.1 gegeben.

Im Falle des Comet Prozessors ist auffallend, dass die Toolchain 291 Register findet, was deutlich über den Erwartungen liegt. Grund dafür ist, dass das HLS Tool die 32-Bit Register in 16-Bit und zwei 8-Bit Register aufteilt. Dies führt folglich zu einer deutlich höheren Anzahl an erkannten Registern. Darüber hinaus fällt auf, dass der Registersatz aus nur 31 anstatt 32 Registern besteht, was hier erneut an dem HLS Tool liegt, da dieses ein Register aus dem C Code

Architektur	Comet [262]	PicoRV32 [264]	tinyriscv [265]
Anzahl FFs	1694	1692	1956
Erkannte Register	291	87	74
Erkannte Zähler	3	4	2
Erkannte RF-Register	31	32	38
Zustandserhaltende FFs	1030	1185	1312

Tabelle 7.1: Ergebnisse der Toolchain für drei Zielanwendungen

nicht in Verilog überführt. Die 31 Register werden dabei korrekt von der vorgestellten Toolchain erkannt, genauso wie der 30-bit Befehlszähler. Folglich wurden alle zustandserhaltenden FFs in der Comet Architektur korrekt erkannt und im weiteren Schritt durch hybride MFFs ersetzt.

Daneben markiert die Toolchain zwei Pipeline-Register für den Opcode in der Decode- und der Execute-Stage als 5-Bit bzw. 3-Bit Zähler, welche nicht für Checkpointing benötigt werden. Folglich werden acht FFs mehr als notwendig durch hybride MFFs ersetzt, was bei einer Gesamtzahl von 1030 erkannten zustandserhaltenden FFs vernachlässigbar wenig ist. Insgesamt wurden folglich etwa 61 % der FFs in der Architektur des Comet Prozessors als zustandserhaltend markiert. Der relative hohe Prozentsatz folgt aus dem minimalen Design des Kerns, welcher über wenige zusätzliche Komponenten verfügt.

Für den PicoRV32 ergibt die Synthese eine Anzahl von 1692 FFs für das gesamte Design. Darin erkennt der vorgeschlagene Algorithmus in der Registergruppierung 87 Register, wovon drei im nächsten Schritt als Zähler erkannt werden. Dabei handelt es sich um den Befehlszähler sowie einen internen 64-Bit Taktzähler und einen 64-Bit Instruktionszähler. Diese sind jeweils für Checkpointing erforderlich, folglich funktioniert die Toolchain korrekt. Neben den 32 32-Bit Registern des Registersatzes, welche ebenfalls vom Algorithmus erkannt werden, markiert dieser fälschlicherweise ein FF des Speicherinterfaces als zustandserhaltend. Dennoch weisen auch die Ergebnisse für den PicoRV32 nach, dass alle für das Checkpointing relevanten FFs korrekt identifiziert wurden und die Anzahl der falsch erkannten Register, in diesem Fall eins, sehr gering ausfällt. Für den PicoRV32 ergeben sich folglich 1185 zustandserhaltende FFs, also ca. 70 %, welche im nächsten Schritt von der Toolchain durch hybride MFFs ersetzt werden.

Wie im Falle der zuvor analysierten Prozessorkerne, identifiziert die Toolchain auch für den tinyriscv den gesamten Registersatz sowie den Befehls-

zähler und einen 64-Bit Taktzähler korrekt als zustandserhaltend. Darüber hinaus werden alle fünf Status- und Steuerregister in den 1956 FFs des tinyriscv erkannt. Die Toolchain markiert lediglich das Ergebnisregister des Dividers fälschlicherweise, wodurch auch für diesen Prozessor der unnötige Overhead sehr klein ausfällt. Insgesamt werden 1312 FFs als zustandserhaltend von der Toolchain identifiziert, was einem Prozentsatz von 67 % entspricht.

Abschließend soll kurz auf den daraus resultierenden zusätzlichen Flächenbedarf durch das Ersetzen der FFs durch hybride MFFs eingegangen werden. Basierend auf der Annahme in Abschnitt 7.4, in dem von einem Flächenverbrauch von 300 % im Vergleich zu CMOS FFs ausgegangen wird, ergibt sich für Comet, PicoRV32 und tinyriscv eine Änderung um 183 %, 210 % sowie 201 %. Allerdings ziehen diese Zahlen lediglich den Prozessor selbst in Betracht, weshalb der Overhead bei Einbeziehung von Speichern und anderen Komponenten in einer Mikrocontrollerarchitektur deutlich geringer ausfällt. Darüber hinaus können auch bei der Verdrahtung der hybriden MFFs weitere Optimierungen vorgenommen werden, da die Abschätzungen des Flächenbedarfs davon ausgehen, dass jedes hybride MFF über einen eigenen Lese- und Schreibschaltkreis für den integrierten MTK verfügt. Ein Register kann jedoch durch Verwendung von Multi-Bit Strukturen, wie beispielsweise von Chabi et al. vorgeschlagen [256], auf eine einzige Lese- und Schreiblogik reduziert werden. Infolgedessen lässt sich der zusätzliche Flächenbedarf durch den Einsatz hybrider MFFs mittels solcher Optimierungen deutlich verringern.

Kapitel 8

Zusammenfassung und Ausblick

Eingebettete Systeme, wie diese beispielsweise in Fahrzeugen oder der Robotik zu finden sind, unterliegen bestimmten funktionalen und nichtfunktionalen Anforderungen, denen sie genügen müssen. Deshalb bedarf es bereits in der Entwurfsphase entsprechender Simulationsumgebungen und Werkzeuge, die diese Randbedingungen in die Suche nach einer optimalen Systemarchitektur einbeziehen. Die Aufteilung der Arbeitslast über mehrere (heterogene) Recheneinheiten ist eine bekannte Möglichkeit, um die Performance und gegebenenfalls auch die Energieeffizienz zu steigern. Existierende Ansätze lassen sich allerdings nur bedingt auf die DNN-Inferenz anwenden, da diese im Gegensatz zu vielen anderen Anwendungen wesentlich durch den Datenfluss definiert ist. Somit bedarf es neuer Lösungsansätze im Rahmen eines umfassenden Hardware/Software Co-Designs, um die relevanten Metriken eines Systems für die Inferenz-Partitionierung von DNNs zu optimieren.

Zusammenfassung

Die vorliegende Arbeit stellt hierfür ein Konzept auf unterschiedlichen Entwurfsebenen vor, welches die Systemarchitektur hinsichtlich verschiedener Optimierungsziele untersucht und somit den Entwurfsprozess unterstützt. Der thematische Schwerpunkt liegt dabei auf der Inferenz-Partitionierung von DNNs auf Systemebene in Kapitel 5, welche zunächst in zwei Vorstudien untersucht wurde. Diese konnten einerseits für ein eingebettetes System, bestehend aus einem reinen Hardwarebeschleuniger und einer im Handel verfügbaren Plattform, sowie für ein FPGA-basiertes System nachweisen, dass die Optimierung der Inferenz-Partitionierung hinsichtlich Latenz, Energiebedarf, Link-Bandbreite und Ressourcenbedarf vorteilhaft ist. Für die Exploration eines eingebetteten Systems, bestehend aus Sensorknoten und

zentraler Rechenplattform, wurde ein Framework entwickelt, welches die relevanten Metriken basierend auf Simulationen analytischer Modelle und Messungen realer Hardwarekomponenten ermittelt und anschließend verschiedene Partitionierungen analysiert. Auf diese Weise konnten erste Erkenntnisse hinsichtlich einer wirkungsvollen Methodik erarbeitet und die Auswirkungen der Inferenz-Partitionierung beispielhaft für drei CNNs untersucht werden. Im Rahmen der zweiten Vorstudie untersuchte die Arbeit mittels eines neuartigen Toolflows die Aufteilung der Inferenz zwischen einem eingebetteten FPGA und einer CPU. Aufgrund der geringen Hardwareressourcen des FPGAs zielte diese Exploration dabei auf eine maximale Beschleunigung der Inferenz durch Auslagerung möglichst weniger Layer auf die CPU bei gleichzeitig geringer Bandbreite am Partitionierungspunkt ab. In der Evaluation konnte auch die zweiten Vorstudie zeigen, dass eine ressourcenbasierte Exploration vorteilhaft ist, um eine energieeffiziente und performante Inferenz-Partitionierung zu erreichen. Aufbauend auf den jeweils gewonnenen Erkenntnissen, leitet die vorliegende Arbeit zunächst eine allgemeine Problemdefinition ab, welche die Grundlage für das im Anschluss vorgestellte Framework CNNParted bildet. Dieses ermöglicht die automatisierte Suche nach einem optimalen Partitionierungsschema für verteilte eingebettete Systeme sowie Multi-Beschleuniger Architekturen, wie diese unter anderem in Chiplet-basierten Plattformen zu finden sind. Dafür identifiziert das Framework zunächst eine topologische Sortierung des zu untersuchenden DNNs, bevor es die einzelnen Layer hinsichtlich Latenz und Energiebedarf für die verfügbaren Hardwarebeschleuniger evaluiert. Zusätzlich erfolgt eine Layerrobustheitsanalyse, welche es ermöglicht, für die Genauigkeit des DNNs nachteilige Schemata in der nachfolgenden Exploration auszusortieren und somit den Suchraum einzuschränken. Basierend auf einem evolutionären Ansatz optimiert CNNParted so die Partitionierung sowie das Mapping der einzelnen Partitionen auf die im System verfügbare Hardware. Als Ergebnis liefert das Framework schließlich eine Pareto-Front aus vorteilhaften Partitionierungsschemata hinsichtlich sieben funktionaler und nicht-funktionaler Metriken. Die Evaluation dieser Methodik für verschiedene DNNs ergibt dabei eine Reduktion der Latenz um bis zu 77 % mit einem geringen Verlust der Genauigkeit von etwa 1 %. Zusätzlich zeigen die Ergebnisse einen deutlichen Anstieg der Energieeffizienz im Vergleich zur Ausführung der Netze auf einem einzigen Hardwarebeschleuniger.

Die vorliegende Arbeit demonstriert anschließend, dass eine Inferenz-Partitionierung ohne weitere Optimierungen auf tieferen Entwurfsebenen nicht immer vorteilhaft ist (s. Kapitel 6). Insbesondere Anwendungen im Bereich des IoT, welche für die Kommunikation zwischen den verteilten

Plattformen im System ein energieeffizientes aber langsames Interface nutzen, erfordern im Entwurf ein umfassenderes Hardware/Software Co-Design. Folglich werden verschiedene Ansätze präsentiert, wie eine solche Realisierung vorteilhaft erfolgen kann. Anhand des Beispiels eines LSTM-Beschleunigers namens ATLAS wird die Anwendung von Approximate Computing untersucht. Die Ergebnisse der Evaluation ergaben einen geringen Einfluss von approximativen Multiplizierern und Aktivierungsfunktionen auf die Genauigkeit nach der Durchführung von QAT, während deutliche Einsparungen der Hardwareressourcen zu beobachten waren. Im Gegensatz dazu wurde der anschließend vorgestellte TCN-Beschleuniger LOTTA hinsichtlich der verfügbaren Hardwarekomponenten eines Low-Power-FPGAs entworfen und auf Performance und Energieeffizienz untersucht. Trotz der Verwendung externer Speicher via Quad SPI, welche für den Großteil des Energiebedarfs verantwortlich sind, ergab die Evaluation des Gesamtsystems eine Leistungsaufnahme von lediglich 27,81 mW. Mittels einer entsprechenden Hyperparametersuche für ein TCN konnte die vorliegende Arbeit abschließend beispielhaft anhand der Trägheitsnavigation und Spracherkennung die Anwendbarkeit dieser Systemarchitektur für den Einsatz im IoT darlegen.

Zuletzt behandelt Kapitel 7 eine weitere Optimierung auf technologischer Ebene, welche sicherheitskritische Anwendungen betrifft. In einem solchen Umfeld müssen robuste Systeme eingesetzt werden, um Systemfehler, einschließlich Stromausfälle, auszugleichen. Hierfür stellt die vorliegende Arbeit eine analytische Methodik vor, um alle zustandserhaltenden FFs in einer Netzliste zu finden und diese durch hybride MFFs zu ersetzen. Auf diese Weise ist das Erstellen von Checkpoints und die Durchführung von Rollbacks ohne die Notwendigkeit der redundanten Auslegung ganzer Systeme möglich. Die Evaluationsergebnisse konnten demonstrieren, dass die implementierten Algorithmen in der Lage sind, alle zustandserhaltenden FFs für drei RISC-V-basierte Prozessorkerne zu erkennen.

Ausblick

Im Mittelpunkt der vorliegenden Arbeit steht die Inferenz-Partitionierung von vortrainierten DNNs in der Entwurfsphase des Systems. Insbesondere aus den Ergebnissen der Exploration mittels CNNParted können wichtige Erkenntnisse für die Hard- und Softwarearchitektur des Systems gewonnen und für den Einsatz in realen Umgebungen genutzt werden. Dennoch bildet

die hier vorgestellte Methodik nicht den kompletten Zyklus der Entwicklung eingebetteter Systeme für ML ab. Eine Einbeziehung der Partitionierung in die NAS erscheint sinnvoll, um die Grundstruktur des DNNs derart zu gestalten, dass die Aufteilung des Netzes an Partitionierungspunkten vorgenommen wird, welche nur geringe Anforderungen an die Linkbandbreite haben. Dies ist besonders für Anwendungen im IoT sinnvoll, um drahtlose Schnittstellen wie BLE möglichst wenig auszulasten und gleichzeitig geringe Latenzen zu erzielen. Aber auch heterogene Systeme, bestehend aus analogen und digitalen Beschleunigern, könnten davon profitieren, wenn die Auswirkungen der Quantisierung sowie die Fehleranfälligkeit analoger Komponenten bereits während der NAS mit in die Entscheidungsfindung einbezogen würden. Existierende Ansätze einer solchen NAS betrachten nicht alle relevanten Metriken und sind häufig auf einen speziellen Anwendungsfall optimiert. Sie können daher nicht direkt in einen Toolflow mit CNNParted oder ähnlichen Methoden eingebunden werden.

Im Rahmen der wissenschaftlichen Arbeit [Sch25] wurde in einem ersten Schritt untersucht, wie eine Entwurfsraumexploration (DSE) der Hardwarearchitekturen als Teil von CNNParted in die Inferenz-Partitionierung von DNNs integriert werden kann. Der Algorithmus evaluiert zunächst verschiedene Konfigurationen der digitalen Beschleuniger, um Pareto-optimale Punkte hinsichtlich Latenz, Energie- und Flächenbedarf zu ermitteln. Die Exploration führt zu einer begrenzten Anzahl potenzieller Architekturen für die jeweilige Hardwareplattform, welche im Anschluss in die Suche nach vorteilhaften Partitionierungsschemata einbezogen werden. Wie in Tabelle 8.1 dargestellt, resultiert dies in einer Verbesserung relevanter Metriken im Vergleich zur Exploration mit fixen Hardwarearchitekturen, wobei sich die Laufzeit des Frameworks lediglich marginal erhöht. Die Ergebnisse dieser ersten Untersuchung belegen folglich, dass eine Einbeziehung der DSE in die Partitionierung vorteilhaft und grundsätzlich realisierbar ist.

Darüber hinaus kann neben der Betrachtung der Inferenz auch die Einbeziehung des Trainings im Hinblick auf das Partitionierungsschema sinnvoll sein. Insbesondere eingebettete Systeme im IoT sind in ihrer Leistungsaufnahme stark limitiert und erlauben daher kein lokales Training. Um dennoch eine Anpassung der Gewichte zur Verbesserung der Genauigkeit zur Laufzeit zu ermöglichen, ist es denkbar, nur die in der zentralen Rechenplattform ausgeführten Layer für die Feinabstimmung des Netzes zu berücksichtigen. Um eine ganzheitliche Partitionierung von DNNs zu realisieren, sind daher weitere Untersuchungen hinsichtlich des Einflusses einzelner Schichten auf das Gesamtergebnis des Netzes notwendig. Ein erster Schritt in diese Richtung erfolgte bereits im Rahmen der Untersuchung von Transfer Learning

Architektur	Variante	d / ms	e / mJ	s / mm²
GoogLeNet	Latenz	30.05	7.91	13.81
	Energie	30.05	7.91	13.81
GoogLeNet DSE	Latenz	23.28	8.02	18.97
	Energie	30.05	7.91	12.72
ResNeXt-50	Latenz	113.30	30.01	13.81
	Energie	113.30	30.01	13.81
ResNeXt-50 DSE	Latenz	90.10	39.11	14.27
	Energie	113.30	30.01	12.72
Yolo v5	Latenz	12.91	3.67	13.81
	Energie	13.84	3.59	13.81
Yolo v5 DSE	Latenz	12.37	3.62	14.31
	Energie	13.84	3.59	12.72

Tabelle 8.1: Ausgewählte Ergebnisse der Entwurfsraumexploration von Beschleunigerarchitekturen als Teil von CNNParted aus [Sch25]

im Kontext der Partitionierung [Top25]. Die in Abschnitt 5.4.3 präsentierte Robustheitsanalyse dient dem Algorithmus als Grundlage für die Identifikation eines vorteilhaften Partitionierungspunktes. Im Rahmen dessen wird eine Maximierung der Anzahl an 8-Bit-Integer quantisierter Schichten sowie der Top-1-Genauigkeit angestrebt, während gleichzeitig eine Minimierung der Rechengenauigkeit erfolgt, um ein energieeffizientes lokales Training zu ermöglichen. Das ermittelte Partitionierungsschema umfasst die unteren, 8-Bit-Integer quantisierten Schichten, sowie die oberen Schichten, die mit bfloat16 quantisiert sind. Die Ergebnisse der Untersuchung weisen nach, dass der Algorithmus eine vorteilhafte Partitionierung findet, ohne dass dies zu einer signifikanten Abnahme der Genauigkeit im Vergleich zur Basisgenauigkeit der Modelle in bfloat16 führt. Somit lässt sich der Energiebedarf für das Re-Training auf einen neuen Datensatz deutlich reduzieren. Die Abbildung 8.1 veranschaulicht die Auswirkung der Partitionierung auf die Genauigkeit des Netzes nach dem Transfer-Learning im Vergleich zu nicht partitionierten Modellen. Exemplarisch sind die Ergebnisse der Exploration für ResNet-18 und SqueezeNet V1.1 dargestellt, welche mittels ImageNet trainiert wurden. Diese Resultate demonstrieren, dass neben der Inferenz auch das Training von einer Partitionierung profitiert. Im Rahmen der ganzheitlichen Partitionierung von DNNs ist es in einem nächsten Schritt erforderlich, die vorliegenden Ergebnisse mit den Charakteristika relevanter Hardwarearchitekturen zu verknüpfen, um vorteilhafte Aufteilungen der Netze für das Training in eingebetteten Systemen zu identifizieren.

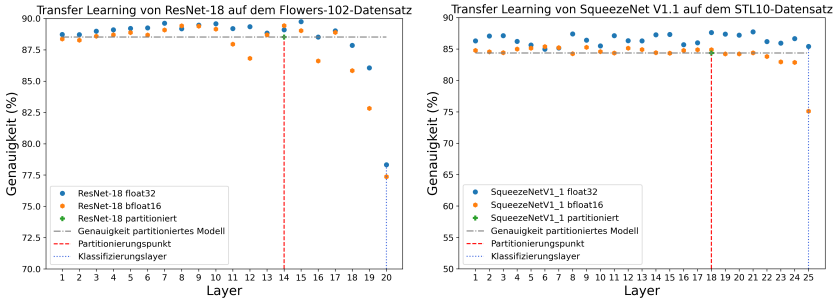


Abbildung 8.1: Ausgewählte Ergebnisse der Untersuchung von Partitionierung für Transfer Learning aus [Top25]

Die dynamische Inferenz-Partitionierung zur Systemlaufzeit ist ein weiterer Aspekt, der in dieser Arbeit nicht berücksichtigt wurde. Wie in der Arbeit [Sta23] demonstriert, können die Ergebnisse von CNNParted zwar auch für dynamisches Scheduling paralleler KI-Anwendungen zum Einsatz kommen, jedoch liefert dieser Ansatz in der Regel keine optimalen Ergebnisse. Grund dafür ist, dass die Problemstellung hier eine andere ist. Um ein optimales Scheduling paralleler DNNs hinsichtlich verschiedener Metriken zu erreichen, können auch nicht-optimale Lösungen der Inferenz-Partitionierung eines Netzes relevant sein. Daher ist es notwendig, diese Randbedingung bereits in die Exploration valider Partitionierungsschemata einzubeziehen.

Literatur

[1] Zhao, Wayne Xin u. a.: A Survey of Large Language Models. 2023. arXiv: 2303.18223 [cs.CL]. URL: https://arxiv.org/abs/2303.18223.

[2] Yadav, Samir S. und Jadhav, Shivajirao M.: „Deep convolutional neural network based medical image classification for disease diagnosis". In: *Journal of Big Data* 6.1 (Dez. 2019), S. 113. DOI: 10.1186/s40537-019-0276-2. URL: https://doi.org/10.1186/s40537-019-0276-2.

[3] Liao, Yiyi; Xie, Jun und Geiger, Andreas: „KITTI-360: A Novel Dataset and Benchmarks for Urban Scene Understanding in 2D and 3D". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.3 (2023), S. 3292–3310. DOI: 10.1109/TPAMI.2022.3179507.

[4] Ottenhaus, Simon; Renninghoff, Daniel; Grimm, Raphael; Ferreira, Fabio und Asfour, Tamim: „Visuo-Haptic Grasping of Unknown Objects based on Gaussian Process Implicit Surfaces and Deep Learning". In: *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*. 2019, S. 402–409. DOI: 10.1109/Humanoids43949.2019.9035002.

[5] Zonta, Tiago; da Costa, Cristiano André; da Rosa Righi, Rodrigo; de Lima, Miromar José; da Trindade, Eduardo Silveira und Li, Guann Pyng: „Predictive maintenance in the Industry 4.0: A systematic literature review". In: *Computers & Industrial Engineering* 150 (2020), S. 106889. DOI: https://doi.org/10.1016/j.cie.2020.106889. URL: https://www.sciencedirect.com/science/article/pii/S0360835220305787.

[6] Bolhasani, Hamidreza; Mohseni, Maryam und Rahmani, Amir Masoud: „Deep learning applications for IoT in health care: A systematic review". In: *Informatics in Medicine Unlocked* 23 (2021), S. 100550. DOI: https://doi.org/10.1016/j.imu.2021.100550. URL: https://www.sciencedirect.com/science/article/pii/S235291482100040X.

[7] Samek, Wojciech; Montavon, Grégoire; Lapuschkin, Sebastian; Anders, Christopher J. und Müller, Klaus-Robert: „Explaining Deep Neural Networks and Beyond: A Review of Methods and Applications". In: *Proceedings of the IEEE* 109.3 (2021), S. 247–278. DOI: 10.1109/JPROC.2021.3060483.

[8] Bianco, Simone; Cadene, Remi; Celona, Luigi und Napoletano, Paolo: „Benchmark Analysis of Representative Deep Neural Network Architectures". In: *IEEE Access* 6 (2018), S. 64270–64277. DOI: 10.1109/ACCESS.2018.2877890.

[9] Capra, Maurizio; Bussolino, Beatrice; Marchisio, Alberto; Masera, Guido; Martina, Maurizio und Shafique, Muhammad: „Hardware and Software Optimizations for Accelerating Deep Neural Networks: Survey of Current Trends,

Challenges, and the Road Ahead“. In: *IEEE Access* 8 (2020), S. 225134–225180. DOI: 10.1109/ACCESS.2020.3039858.

[10] GreenWaves Technologies: GAP8. URL: https://greenwaves-technologies.com/gap8_mcu_ai/ (besucht am 02. 09. 2024).

[11] Google LLC: Edge TPU. URL: https://coral.ai/ (besucht am 02. 09. 2024).

[12] NVIDIA Corporation: NVIDIA Jetson. URL: https://developer.nvidia.com/embedded/jetson-modules (besucht am 02. 09. 2024).

[13] Yue, Jinshan u. a.: „14.3 A 65nm Computing-in-Memory-Based CNN Processor with 2.9-to-35.8TOPS/W System Energy Efficiency Using Dynamic-Sparsity Performance-Scaling Architecture and Energy-Efficient Inter/Intra-Macro Data Reuse“. In: *2020 IEEE International Solid-State Circuits Conference - (ISSCC)*. 2020, S. 234–236. DOI: 10.1109/ISSCC19947.2020.9062958.

[14] Prabhu, Kartik u. a.: „CHIMERA: A 0.92-TOPS, 2.2-TOPS/W Edge AI Accelerator With 2-MByte On-Chip Foundry Resistive RAM for Efficient Training and Inference“. In: *IEEE Journal of Solid-State Circuits* 57.4 (2022), S. 1013–1026. DOI: 10.1109/JSSC.2022.3140753.

[15] Garofalo, Angelo; Ottavi, Gianmarco; Conti, Francesco; Karunaratne, Geethan; Boybat, Irem; Benini, Luca und Rossi, Davide: „A Heterogeneous In-Memory Computing Cluster for Flexible End-to-End Inference of Real-World Deep Neural Networks“. In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 12.2 (2022), S. 422–435. DOI: 10.1109/JETCAS.2022.3170152.

[16] Ueyoshi, Kodai u. a.: „DIANA: An End-to-End Energy-Efficient Digital and ANAlog Hybrid Neural Network SoC“. In: *2022 IEEE International Solid-State Circuits Conference (ISSCC)*. Bd. 65. 2022, S. 1–3. DOI: 10.1109/ISSCC42614.2022.9731716.

[17] Tan, Zhanhong; Cai, Hongyu; Dong, Runpei und Ma, Kaisheng: „NN-Baton: DNN Workload Orchestration and Chiplet Granularity Exploration for Multi-chip Accelerators“. In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 2021, S. 1013–1026. DOI: 10.1109/ISCA52012.2021.00083.

[18] Kao, Sheng-Chun und Krishna, Tushar: „MAGMA: An Optimization Framework for Mapping Multiple DNNs on Multiple Accelerator Cores“. In: *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 2022, S. 814–830. DOI: 10.1109/HPCA53966.2022.00065.

[19] Gholami, Amir; Yao, Zhewei; Kim, Sehoon; Hooper, Coleman; Mahoney, Michael W. und Keutzer, Kurt: AI and Memory Wall. 2024. arXiv: 2403.14123 [cs.LG]. URL: https://arxiv.org/abs/2403.14123.

[20] Bringmann, Oliver; Lange, Walter und Bogdan, Martin: Eingebettete Systeme. 4. überarbeitete Auflage. De Gruyter Studium. Berlin: De Gruyter Oldenbourg, 2022. URL: https://doi.org/10.1515/9783110702064.

[21] Kelleher, John D: Deep learning. MIT press, 2019.

[22] LeCun, Yann u. a.: „Generalization and network design strategies“. In: *Connectionism in perspective* 19.143-155 (1989), S. 18.

[23] León, Fernando Puente und Jäkel, Holger: Signale und Systeme. Berlin, Boston: De Gruyter Oldenbourg, 2019. DOI: doi:10.1515/9783110626322. URL: https://doi.org/10.1515/9783110626322.

[24] Goodfellow, Ian; Bengio, Yoshua und Courville, Aaron: Deep Learning. http://www.deeplearningbook.org. MIT Press, 2016.

[25] Long, Jonathan; Shelhamer, Evan und Darrell, Trevor: „Fully Convolutional Networks for Semantic Segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Juni 2015.

[26] Ma, Wei und Lu, Jun: An Equivalence of Fully Connected Layer and Convolutional Layer. 2017. arXiv: 1712.01252 [cs.LG]. URL: https://arxiv.org/abs/1712.01252.

[27] Hochreiter, Sepp und Schmidhuber, Jürgen: „Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), S. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[28] Christopher Olah: Understanding LSTM Networks. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (besucht am 26. 08. 2024).

[29] Bruneo, Dario und De Vita, Fabrizio: „On the Use of LSTM Networks for Predictive Maintenance in Smart Industries". In: *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*. 2019, S. 241–248. DOI: 10.1109/SMARTCOMP.2019.00059.

[30] Graves, Alex und Jaitly, Navdeep: „Towards End-To-End Speech Recognition with Recurrent Neural Networks". In: *Proceedings of the 31st International Conference on Machine Learning*. Hrsg. von Xing, Eric P. und Jebara, Tony. Bd. 32. Proceedings of Machine Learning Research 2. Bejing, China: PMLR, 22–24 Jun 2014, S. 1764–1772. URL: https://proceedings.mlr.press/v32/graves14.html.

[31] Oord, Aaron van den; Dieleman, Sander; Zen, Heiga; Simonyan, Karen; Vinyals, Oriol; Graves, Alex; Kalchbrenner, Nal; Senior, Andrew und Kavukcuoglu, Koray: WaveNet: A Generative Model for Raw Audio. 2016. arXiv: 1609.03499 [cs.SD]. URL: https://arxiv.org/abs/1609.03499.

[32] Nan, Mihai; Trăscău, Mihai; Florea, Adina Magda und Iacob, Cezar Cătălin: „Comparison between Recurrent Networks and Temporal Convolutional Networks Approaches for Skeleton-Based Action Recognition". In: *Sensors* 21.6 (2021). DOI: 10.3390/s21062051. URL: https://www.mdpi.com/1424-8220/21/6/2051.

[33] Remy, Philippe: Temporal Convolutional Networks for Keras. 2020. (Besucht am 26. 08. 2024).

[34] Novac, Pierre-Emmanuel; Boukli Hacene, Ghouthi; Pegatoquet, Alain; Miramond, Benoît und Gripon, Vincent: „Quantization and Deployment of Deep Neural Networks on Microcontrollers". In: *Sensors* 21.9 (2021). DOI: 10.3390/s21092984. URL: https://www.mdpi.com/1424-8220/21/9/2984.

[35] Gholami, Amir; Kim, Sehoon; Dong, Zhen; Yao, Zhewei; Mahoney, Michael W. und Keutzer, Kurt: A Survey of Quantization Methods for Efficient Neural Network Inference. 2021. arXiv: 2103.13630 [cs.CV].

[36] Bengio, Yoshua; Léonard, Nicholas und Courville, Aaron: Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. 2013. arXiv: 1308.3432 [cs.LG]. URL: https://arxiv.org/abs/1308.3432.

[37] Banner, Ron; Nahshan, Yury und Soudry, Daniel: „Post training 4-bit quantization of convolutional networks for rapid-deployment". In: *Advances in Neural Information Processing Systems*. Hrsg. von Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E. und Garnett, R. Bd. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/c0a62e133894cdce435bcb4a5df1db2d-Paper.pdf.

[38] Cao, Qingqing; Irimiea, Alexandru E.; Abdelfattah, Mohamed; Balasubramanian, Aruna und Lane, Nicholas D.: „Are Mobile DNN Accelerators Accelerating DNNs?" In: *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*. EMDL'21. Virtual, WI, USA: Association for Computing Machinery, 2021, S. 7–12. DOI: 10.1145/3469116.3470011. URL: https://doi.org/10.1145/3469116.3470011.

[39] Zhang, Chen; Li, Peng; Sun, Guangyu; Guan, Yijin; Xiao, Bingjun und Cong, Jason: „Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks". In: *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. FPGA '15. Monterey, California, USA: Association for Computing Machinery, 2015, S. 161–170. DOI: 10.1145/2684746.2689060. URL: https://doi.org/10.1145/2684746.2689060.

[40] Shao, Yakun Sophia u. a.: „Simba: Scaling Deep-Learning Inference with Multi-Chip-Module-Based Architecture". In: *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO '52. Columbus, OH, USA: Association for Computing Machinery, 2019, S. 14–27. DOI: 10.1145/3352460.3358302. URL: https://doi.org/10.1145/3352460.3358302.

[41] Genc, Hasan; Haj-Ali, Ameer; Iyer, Vighnesh; Amid, Alon; Mao, Howard; Wright, John; Schmidt, Colin; Zhao, Jerry; Ou, Albert; Banister, Max u. a.: „Gemmini: An agile systolic array generator enabling systematic evaluations of deep-learning architectures". In: *arXiv preprint arXiv:1911.09925* 3 (2019).

[42] Dhilleswararao, Pudi; Boppu, Srinivas; Manikandan, M. Sabarimalai und Cenkeramaddi, Linga Reddy: „Efficient Hardware Architectures for Accelerating Deep Neural Networks: Survey". In: *IEEE Access* 10 (2022), S. 131788–131828. DOI: 10.1109/ACCESS.2022.3229767.

[43] Cavigelli, Lukas; Gschwend, David; Mayer, Christoph; Willi, Samuel; Muheim, Beat und Benini, Luca: „Origami: A Convolutional Network Accelerator". In: *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*. GLSVLSI '15. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 2015, S. 199–204. DOI: 10.1145/2742060.2743766. URL: https://doi.org/10.1145/2742060.2743766.

[44] Du, Zidong; Fasthuber, Robert; Chen, Tianshi; Ienne, Paolo; Li, Ling; Luo, Tao; Feng, Xiaobing; Chen, Yunji und Temam, Olivier: „ShiDianNao: Shifting vision processing closer to the sensor". In: *2015 ACM/IEEE 42nd Annual Inter-*

national Symposium on Computer Architecture (ISCA). 2015, S. 92–104. DOI: 10.1145/2749469.2750389.

[45] Sim, Jaehyeong; Lee, Somin und Kim, Lee-Sup: „An Energy-Efficient Deep Convolutional Neural Network Inference Processor With Enhanced Output Stationary Dataflow in 65-nm CMOS“. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28.1 (2020), S. 87–100. DOI: 10.1109/TVLSI.2019.2935251.

[46] Chen, Tianshi; Du, Zidong; Sun, Ninghui; Wang, Jia; Wu, Chengyong; Chen, Yunji und Temam, Olivier: „DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning“. In: *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '14. Salt Lake City, Utah, USA: Association for Computing Machinery, 2014, S. 269–284. DOI: 10.1145/2541940.2541967. URL: https://doi.org/10.1145/2541940.2541967.

[47] Chen, Yu-Hsin; Yang, Tien-Ju; Emer, Joel und Sze, Vivienne: „Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices“. In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9.2 (2019), S. 292–308. DOI: 10.1109/JETCAS.2019.2910232.

[48] Chen, Yu-Hsin; Krishna, Tushar; Emer, Joel S. und Sze, Vivienne: „Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks“. In: *IEEE Journal of Solid-State Circuits* 52.1 (2017), S. 127–138. DOI: 10.1109/JSSC.2016.2616357.

[49] Shafiee, Ali; Nag, Anirban; Muralimanohar, Naveen; Balasubramonian, Rajeev; Strachan, John Paul; Hu, Miao; Williams, R. Stanley und Srikumar, Vivek: „ISAAC: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars“. In: *Proceedings of the 43rd International Symposium on Computer Architecture*. ISCA '16. Seoul, Republic of Korea: IEEE Press, 2016, S. 14–26. DOI: 10.1109/ISCA.2016.12. URL: https://doi.org/10.1109/ISCA.2016.12.

[50] Chi, Ping; Li, Shuangchen; Xu, Cong; Zhang, Tao; Zhao, Jishen; Liu, Yongpan; Wang, Yu und Xie, Yuan: „PRIME: a novel processing-in-memory architecture for neural network computation in ReRAM-based main memory“. In: *Proceedings of the 43rd International Symposium on Computer Architecture*. ISCA '16. Seoul, Republic of Korea: IEEE Press, 2016, S. 27–39. DOI: 10.1109/ISCA.2016.13. URL: https://doi.org/10.1109/ISCA.2016.13.

[51] Eldebiky, Amro; Zhang, Grace Li; Böcherer, Georg; Li, Bing und Schlichtmann, Ulf: „CorrectNet+: Dealing With HW Non-Idealities in In-Memory-Computing Platforms by Error Suppression and Compensation“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43.2 (2024), S. 573–585. DOI: 10.1109/TCAD.2023.3313089.

[52] Kwon, Hyoukjun; Chatarasi, Prasanth; Sarkar, Vivek; Krishna, Tushar; Pellauer, Michael und Parashar, Angshuman: „MAESTRO: A Data-Centric Approach to Understand Reuse, Performance, and Hardware Cost of DNN Mappings“. In: *IEEE Micro* 40.3 (2020), S. 20–29. DOI: 10.1109/MM.2020.2985963.

[53] Kwon, Hyoukjun; Samajdar, Ananda und Krishna, Tushar: „MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects“. In: *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '18. Williamsburg, VA, USA: Association for Computing Machinery, 2018, S. 461–475. doi: 10.1145/3173162.3173176. url: https://doi.org/10.1145/3173162.3173176.

[54] Aghli, Nima und Ribeiro, Eraldo: „Combining Weight Pruning and Knowledge Distillation for CNN Compression“. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. Juni 2021, S. 3191–3198.

[55] Domschke, Wolfgang und Drexl, Andreas: Einführung in Operations Research. 9. überarb. u. verb. Aufl. 2015. SpringerLink. Berlin: Springer Gabler, 2015. url: http://dx.doi.org/10.1007/978-3-662-48216-2.

[56] Nickel, Stefan und Rebennack, Steffen: Operations Research. 3. überarbeitete und erweiterte Auflage. Springer eBook Collection. Berlin: Springer Gabler, 2022. url: https://doi.org/10.1007/978-3-662-65346-3.

[57] Kruse, Rudolf und Borgelt, Christian: Computational Intelligence. 2. Aufl. 2015. Computational Intelligence. Wiesbaden: Springer Vieweg, 2015. url: http://dx.doi.org/10.1007/978-3-658-10904-2.

[58] Srinivas, N. und Deb, Kalyanmoy: „Muiltiobjective Optimization Using Nondominated Sorting in Genetic Algorithms“. In: *Evolutionary Computation* 2.3 (1994), S. 221–248. doi: 10.1162/evco.1994.2.3.221.

[59] Deb, K.; Pratap, A.; Agarwal, S. und Meyarivan, T.: „A fast and elitist multiobjective genetic algorithm: NSGA-II“. In: *IEEE Transactions on Evolutionary Computation* 6.2 (2002), S. 182–197. doi: 10.1109/4235.996017.

[60] Chang, Yijia; Huang, Xi; Shao, Ziyu und Yang, Yang: „An Efficient Distributed Deep Learning Framework for Fog-Based IoT Systems“. In: *2019 IEEE Global Communications Conference (GLOBECOM)*. 2019, S. 1–6. doi: 10.1109/GLOBECOM38437.2019.9014056.

[61] Hsu, Chia-Chun; Yang, Chung-Kai; Kuo, Jian-Jhih; Chen, Wen-Tsuen und Sheu, Jang-Ping: „Cooperative Convolutional Neural Network Deployment over Mobile Networks“. In: *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*. 2020, S. 1–7. doi: 10.1109/ICC40277.2020.9149094.

[62] Hu, Chenghao und Li, Baochun: „Distributed Inference with Deep Learning Models across Heterogeneous Edge Devices“. In: *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*. 2022, S. 330–339. doi: 10.1109/INFOCOM48880.2022.9796896.

[63] Baccour, Emna; Erbad, Aiman; Mohamed, Amr; Hamdi, Mounir und Guizani, Mohsen: „DistPrivacy: Privacy-Aware Distributed Deep Neural Networks in IoT surveillance systems“. In: *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*. 2020, S. 1–6. doi: 10.1109/GLOBECOM42002.2020.9322470.

[64] Dey, Swarnava; Mukherjee, Arijit; Pal, Arpan und Balamuralidhar, P.: „Partitioning of CNN Models for Execution on Fog Devices“. In: *Proceedings of the 1st ACM International Workshop on Smart Cities and Fog Computing*. CitiFog'18. Shenzhen, China: Association for Computing Machinery, 2018, S. 19–24. doi: 10.1145/3277893.3277899. url: https://doi.org/10.1145/3277893.3277899.

[65] Jiang, Weiwen; Sha, Edwin H.-M.; Zhang, Xinyi; Yang, Lei; Zhuge, Qingfeng; Shi, Yiyu und Hu, Jingtong: „Achieving Super-Linear Speedup across Multi-FPGA for Real-Time DNN Inference“. In: *ACM Trans. Embed. Comput. Syst.* 18.5s (Okt. 2019). doi: 10.1145/3358192. url: https://doi.org/10.1145/3358192.

[66] Hadidi, Ramyad; Cao, Jiashen; Ryoo, Michael S. und Kim, Hyesoon: „Toward Collaborative Inferencing of Deep Neural Networks on Internet-of-Things Devices“. In: *IEEE Internet of Things Journal* 7.6 (2020), S. 4950–4960. doi: 10.1109/JIOT.2020.2972000.

[67] Teerapittayanon, Surat; McDanel, Bradley und Kung, H.T.: „Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices“. In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 2017, S. 328–339. doi: 10.1109/ICDCS.2017.226.

[68] Fang, Yihao; Jin, Ziyi und Zheng, Rong: „TeamNet: A Collaborative Inference Framework on the Edge“. In: *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. 2019, S. 1487–1496. doi: 10.1109/ICDCS.2019.00148.

[69] Mao, Jiachen; Chen, Xiang; Nixon, Kent W.; Krieger, Christopher und Chen, Yiran: „MoDNN: Local distributed mobile computing system for Deep Neural Network“. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. 2017, S. 1396–1401. doi: 10.23919/DATE.2017.7927211.

[70] Zhang, Shuai; Zhang, Sheng; Qian, Zhuzhong; Wu, Jie; Jin, Yibo und Lu, Sanglu: „DeepSlicing: Collaborative and Adaptive CNN Inference With Low Latency“. In: *IEEE Transactions on Parallel and Distributed Systems* 32.9 (2021), S. 2175–2187. doi: 10.1109/TPDS.2021.3058532.

[71] Hou, Xueyu; Guan, Yongjie; Han, Tao und Zhang, Ning: „DistrEdge: Speeding up Convolutional Neural Network Inference on Distributed Edge Devices“. In: *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2022, S. 1097–1107. doi: 10.1109/IPDPS53621.2022.00110.

[72] Fang, Weiwei; Xu, Wenyuan; Yu, Chongchong und Xiong, Neal. N.: „Joint Architecture Design and Workload Partitioning for DNN Inference on Industrial IoT Clusters“. In: *ACM Trans. Internet Technol.* 23.1 (Feb. 2023). doi: 10.1145/3551638. url: https://doi.org/10.1145/3551638.

[73] Jia, Fucheng; Zhang, Deyu; Cao, Ting; Jiang, Shiqi; Liu, Yunxin; Ren, Ju und Zhang, Yaoxue: „CoDL: efficient CPU-GPU co-execution for deep learning inference on mobile devices“. In: *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*. MobiSys '22. Portland, Oregon: Association for Computing Machinery, 2022, S. 209–221. doi: 10.1145/3498361.3538932. url: https://doi.org/10.1145/3498361.3538932.

[74] Peccia, Federico Nicolás; Viehl, Alexander und Bringmann, Oliver: „DIAPASON: Differentiable Allocation, Partitioning and Fusion of Neural Networks for Distributed Inference“. In: *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2024, S. 1–6. doi: 10.23919/DATE58400.2024.10546603.

[75] Zhao, Zhuoran; Barijough, Kamyar Mirzazad und Gerstlauer, Andreas: „DeepThings: Distributed Adaptive Deep Learning Inference on Resource-Constrained IoT Edge Clusters“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.11 (2018), S. 2348–2359. doi: 10.1109/TCAD.2018.2858384.

[76] Zhang, Sai Qian; Lin, Jieyu und Zhang, Qi: „Adaptive Distributed Convolutional Neural Network Inference at the Network Edge with ADCNN“. In: *Proceedings of the 49th International Conference on Parallel Processing*. ICPP '20. Edmonton, AB, Canada: Association for Computing Machinery, 2020. doi: 10.1145/3404397.3404473. url: https://doi.org/10.1145/3404397.3404473.

[77] Zeng, Liekang; Chen, Xu; Zhou, Zhi; Yang, Lei und Zhang, Junshan: „CoEdge: Cooperative DNN Inference With Adaptive Workload Partitioning Over Heterogeneous Edge Devices“. In: *IEEE/ACM Transactions on Networking* 29.2 (2021), S. 595–608. doi: 10.1109/TNET.2020.3042320.

[78] Naveen, Soumyalatha; Kounte, Manjunath R. und Ahmed, Mohammed Riyaz: „Low Latency Deep Learning Inference Model for Distributed Intelligent IoT Edge Clusters“. In: *IEEE Access* 9 (2021), S. 160607–160621. doi: 10.1109/ACCESS.2021.3131396.

[79] Choi, Kyunghwan; Lee, Seongju; Kang, Beom Woo und Park, Yongjun: „Legion: Tailoring Grouped Neural Execution Considering Heterogeneity on Multiple Edge Devices“. In: *2021 IEEE 39th International Conference on Computer Design (ICCD)*. 2021, S. 383–390. doi: 10.1109/ICCD53106.2021.00067.

[80] Kwon, Dongup; Hur, Suyeon; Jang, Hamin; Nurvitadhi, Eriko und Kim, Jangwoo: „Scalable Multi-FPGA Acceleration for Large RNNs with Full Parallelism Levels“. In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 2020, S. 1–6. doi: 10.1109/DAC18072.2020.9218528.

[81] Agut, David Rodríguez; Tornero, Rafael und Flich, Josè: „Towards Efficient Neural Network Model Parallelism on Multi-FPGA Platforms“. In: *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2023, S. 1–6. doi: 10.23919/DATE56975.2023.10137117.

[82] Bouzidi, Halima; Odema, Mohanad; Ouarnoughi, Hamza; Niar, Smail und Al Faruque, Mohammad Abdullah: „Map-and-Conquer: Energy-Efficient Mapping of Dynamic Neural Nets onto Heterogeneous MPSoCs“. In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 2023, S. 1–6. doi: 10.1109/DAC56929.2023.10247722.

[83] Risso, Matteo; Burrello, Alessio; Sarda, Giuseppe Maria; Benini, Luca; Macii, Enrico; Poncino, Massimo; Verhelst, Marian und Pagliari, Daniele Jahier: „Precision-aware Latency and Energy Balancing on Multi-Accelerator Platforms for DNN Inference“. In: *2023 IEEE/ACM International Symposium*

on Low Power Electronics and Design (ISLPED). 2023, S. 1–6. DOI: 10.1109/ISLPED58423.2023.10244311.

[84] Behnam, Payman; Kamal, Uday und Mukhopadhyay, Saibal: An Algorithm-Hardware Co-design Framework to Overcome Imperfections of Mixed-signal DNN Accelerators. 2022. arXiv: 2208.13896 [cs.AR]. URL: https://arxiv.org/abs/2208.13896.

[85] Dash, Saurabh; Luo, Yandong; Lu, Anni; Yu, Shimeng und Mukhopadhyay, Saibal: „Robust Processing-In-Memory With Multibit ReRAM Using Hessian-Driven Mixed-Precision Computation". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.4 (2022), S. 1006–1019. DOI: 10.1109/TCAD.2021.3078408.

[86] Lane, Nicholas D.; Bhattacharya, Sourav; Georgiev, Petko; Forlivesi, Claudio; Jiao, Lei; Qendro, Lorena und Kawsar, Fahim: „DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices". In: *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. 2016, S. 1–12. DOI: 10.1109/IPSN.2016.7460664.

[87] Jiang, Weiwen; Sha, Edwin Hsing-Mean; Zhuge, Qingfeng; Yang, Lei; Chen, Xianzhang und Hu, Jingtong: „Heterogeneous FPGA-Based Cost-Optimal Design for Timing-Constrained CNNs". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.11 (2018), S. 2542–2554. DOI: 10.1109/TCAD.2018.2857098.

[88] Li, Ruihao; Liu, Ke; Cai, Xiaojun; Zhao, Mengying; John, Lizy K. und Jia, Zhiping: „Improving CNN performance on FPGA clusters through topology exploration". In: *Proceedings of the 36th Annual ACM Symposium on Applied Computing*. SAC '21. Virtual Event, Republic of Korea: Association for Computing Machinery, 2021, S. 126–134. DOI: 10.1145/3412841.3441893. URL: https://doi.org/10.1145/3412841.3441893.

[89] Biookaghazadeh, Saman; Ravi, Pravin Kumar und Zhao, Ming: „Toward Multi-FPGA Acceleration of the Neural Networks". In: *J. Emerg. Technol. Comput. Syst.* 17.2 (Apr. 2021). DOI: 10.1145/3432816. URL: https://doi.org/10.1145/3432816.

[90] Zhang, Xinyi; Hao, Cong; Zhou, Peipei; Jones, Alex und Hu, Jingtong: „H2H: heterogeneous model to heterogeneous system mapping with computation and communication awareness". In: *Proceedings of the 59th ACM/IEEE Design Automation Conference*. DAC '22. San Francisco, California: Association for Computing Machinery, 2022, S. 601–606. DOI: 10.1145/3489517.3530509. URL: https://doi.org/10.1145/3489517.3530509.

[91] Zhang, Wentai; Zhang, Jiaxi; Shen, Minghua; Luo, Guojie und Xiao, Nong: „An Efficient Mapping Approach to Large-Scale DNNs on Multi-FPGA Architectures". In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2019, S. 1241–1244. DOI: 10.23919/DATE.2019.8715174.

[92] Alonso, Tobias; Petrica, Lucian; Ruiz, Mario; Petri-Koenig, Jakoba; Umuroglu, Yaman; Stamelos, Ioannis; Koromilas, Elias; Blott, Michaela und Vissers, Kees: „Elastic-DF: Scaling Performance of DNN Inference in FPGA Clouds

through Automatic Partitioning“. In: *ACM Trans. Reconfigurable Technol. Syst.* 15.2 (Dez. 2021). DOI: 10.1145/3470567. URL: https://doi.org/10.1145/3470567.

[93] Kamath, Akshay Karkal; Abi-Karam, Stefan; Bhat, Ashwin und Hao, Cong: „M5: Multi-modal Multi-task Model Mapping on Multi-FPGA with Accelerator Configuration Search“. In: *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2023, S. 1–6. DOI: 10.23919/DATE56975.2023.10136962.

[94] Gacoin, Victor; Kolar, Anthony; Ren, Chengfang und Guinvarc'h, Regis: „Distributing Deep Neural Networks for Maximising Computing Capabilities and Power Efficiency in Swarm“. In: *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2019, S. 1–5. DOI: 10.1109/ISCAS.2019.8702672.

[95] Zhou, Jingyue; Wang, Yihuai; Ota, Kaoru und Dong, Mianxiong: „AAIoT: Accelerating Artificial Intelligence in IoT Systems“. In: *IEEE Wireless Communications Letters* 8.3 (2019), S. 825–828. DOI: 10.1109/LWC.2019.2894703.

[96] Chen, Jianan; Qi, Qi; Wang, Jingyu; Sun, Haifeng und Liao, Jianxin: „Accelerating DNN Inference by Edge-Cloud Collaboration“. In: *2021 IEEE International Performance, Computing, and Communications Conference (IPCCC)*. 2021, S. 1–7. DOI: 10.1109/IPCCC51483.2021.9679434.

[97] Yang, Cian-You; Kuo, Jian-Jhih; Sheu, Jang-Ping und Zheng, Ke-Jun: „Cooperative Distributed Deep Neural Network Deployment with Edge Computing“. In: *ICC 2021 - IEEE International Conference on Communications*. 2021, S. 1–6. DOI: 10.1109/ICC42927.2021.9500668.

[98] Huang, Yutao; Wang, Feng; Wang, Fangxin und Liu, Jiangchuan: „DeePar: A Hybrid Device-Edge-Cloud Execution Framework for Mobile Deep Learning Applications“. In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2019, S. 892–897. DOI: 10.1109/INFCOMW.2019.8845240.

[99] Kang, Yiping; Hauswald, Johann; Gao, Cao; Rovinski, Austin; Mudge, Trevor; Mars, Jason und Tang, Lingjia: „Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge“. In: *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '17. Xi'an, China: Association for Computing Machinery, 2017, S. 615–629. DOI: 10.1145/3037697.3037698. URL: https://doi.org/10.1145/3037697.3037698.

[100] Hu, Chuang; Bao, Wei; Wang, Dan und Liu, Fengming: „Dynamic Adaptive DNN Surgery for Inference Acceleration on the Edge“. In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. 2019, S. 1423–1431. DOI: 10.1109/INFOCOM.2019.8737614.

[101] Kakolyris, Andreas Kosmas; Katsaragakis, Manolis; Masouros, Dimosthenis und Soudris, Dimitrios: „RoaD-RuNNer: Collaborative DNN partitioning and offloading on heterogeneous edge systems“. In: *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2023, S. 1–6. DOI: 10.23919/DATE56975.2023.10137279.

[102] Ghosh, Soumendu Kumar; Raha, Arnab; Raghunathan, Vijay und Raghunathan, Anand: „PArtNNer: Platform-Agnostic Adaptive Edge-Cloud DNN Partitioning for Minimizing End-to-End Latency“. In: *ACM Trans. Embed. Comput. Syst.* 23.1 (Jan. 2024). doi: 10.1145/3630266. url: https://doi.org/10.1145/3630266.

[103] Chang, Jen-I; Kuo, Jian-Jhih; Lin, Chi-Han; Chen, Wen-Tsuen und Sheu, Jang-Ping: „Ultra-Low-Latency Distributed Deep Neural Network over Hierarchical Mobile Networks“. In: *2019 IEEE Global Communications Conference (GLOBECOM)*. 2019, S. 1–6. doi: 10.1109/GLOBECOM38437.2019.9014122.

[104] Li, En; Zeng, Liekang; Zhou, Zhi und Chen, Xu: „Edge AI: On-Demand Accelerating Deep Neural Network Inference via Edge Computing“. In: *IEEE Transactions on Wireless Communications* 19.1 (2020), S. 447–457. doi: 10.1109/TWC.2019.2946140.

[105] Huang, Zhaowu; Dong, Fang; Shen, Dian; Zhang, Junxue; Wang, Huitian; Cai, Guangxing und He, Qiang: „Enabling Low Latency Edge Intelligence based on Multi-exit DNNs in the Wild“. In: *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. 2021, S. 729–739. doi: 10.1109/ICDCS51616.2021.00075.

[106] Teerapittayanon, Surat; McDanel, Bradley und Kung, H.T.: „BranchyNet: Fast inference via early exiting from deep neural networks“. In: *2016 23rd International Conference on Pattern Recognition (ICPR)*. 2016, S. 2464–2469. doi: 10.1109/ICPR.2016.7900006.

[107] Pacheco, Roberto G. und Couto, Rodrigo S.: „Inference Time Optimization Using BranchyNet Partitioning“. In: *2020 IEEE Symposium on Computers and Communications (ISCC)*. 2020, S. 1–6. doi: 10.1109/ISCC50000.2020.9219647.

[108] Laskaridis, Stefanos; Venieris, Stylianos I.; Almeida, Mario; Leontiadis, Ilias und Lane, Nicholas D.: „SPINN: synergistic progressive inference of neural networks over device and cloud“. In: *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. MobiCom '20. London, United Kingdom: Association for Computing Machinery, 2020. doi: 10.1145/3372224.3419194. url: https://doi.org/10.1145/3372224.3419194.

[109] Zeng, Liekang; Li, En; Zhou, Zhi und Chen, Xu: „Boomerang: On-Demand Cooperative Deep Neural Network Inference for Edge Intelligence on the Industrial Internet of Things“. In: *IEEE Network* 33.5 (2019), S. 96–103. doi: 10.1109/MNET.001.1800506.

[110] Wang, Zhaohao; Zhou, Haochang; Wang, Mengxing; Cai, Wenlong; Zhu, Daoqian; Klein, Jacques-Olivier und Zhao, Weisheng: „Proposal of Toggle Spin Torques Magnetic RAM for Ultrafast Computing“. In: *IEEE Electron Device Letters* 40.5 (2019), S. 726–729. doi: 10.1109/LED.2019.2907063.

[111] Banitalebi-Dehkordi, Amin; Vedula, Naveen; Pei, Jian; Xia, Fei; Wang, Lanjun und Zhang, Yong: „Auto-Split: A General Framework of Collaborative Edge-Cloud AI“. In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. KDD '21. Virtual Event, Singapore: Association

for Computing Machinery, 2021, S. 2543–2553. DOI: 10.1145/3447548.3467078. URL: https://doi.org/10.1145/3447548.3467078.

[112] Ghasemi, Mehdi; Heidari, Soroush; Kim, Young Geun; Lamb, Aaron; Wu, Carole-Jean und Vrudhula, Sarma: „Energy-Efficient Mapping for a Network of DNN Models at the Edge“. In: *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*. 2021, S. 25–30. DOI: 10.1109/SMARTCOMP52413.2021.00024.

[113] Eshratifar, Amir Erfan; Abrishami, Mohammad Saeed und Pedram, Massoud: „JointDNN: An Efficient Training and Inference Engine for Intelligent Mobile Cloud Computing Services“. In: *IEEE Transactions on Mobile Computing* 20.2 (2021), S. 565–576. DOI: 10.1109/TMC.2019.2947893.

[114] Parthasarathy, Arjun und Krishnamachari, Bhaskar: „DEFER: Distributed Edge Inference for Deep Neural Networks“. In: *2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS)*. 2022, S. 749–753. DOI: 10.1109/COMSNETS53615.2022.9668515.

[115] Li, Hongshan; Hu, Chenghao; Jiang, Jingyan; Wang, Zhi; Wen, Yonggang und Zhu, Wenwu: „JALAD: Joint Accuracy-And Latency-Aware Deep Structure Decoupling for Edge-Cloud Execution“. In: *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*. 2018, S. 671–678. DOI: 10.1109/PADSW.2018.8645013.

[116] Almeida, Mario; Laskaridis, Stefanos; Venieris, Stylianos I.; Leontiadis, Ilias und Lane, Nicholas D.: „DynO: Dynamic Onloading of Deep Neural Networks from Cloud to Device“. In: *ACM Trans. Embed. Comput. Syst.* 21.6 (Okt. 2022). DOI: 10.1145/3510831. URL: https://doi.org/10.1145/3510831.

[117] Hao, Zhiwei; Xu, Guanyu; Luo, Yong; Hu, Han; An, Jianping und Mao, Shiwen: „Multi-Agent Collaborative Inference via DNN Decoupling: Intermediate Feature Compression and Edge Learning“. In: *IEEE Transactions on Mobile Computing* 22.10 (2023), S. 6041–6055. DOI: 10.1109/TMC.2022.3183098.

[118] Yao, Shuochao; Li, Jinyang; Liu, Dongxin; Wang, Tianshi; Liu, Shengzhong; Shao, Huajie und Abdelzaher, Tarek: „Deep Compressive Offloading: Speeding up Neural Network Inference by Trading Edge Computation for Network Latency“. In: *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. SenSys '20. Virtual Event, Japan: Association for Computing Machinery, 2020, S. 476–488. DOI: 10.1145/3384419.3430898. URL: https://doi.org/10.1145/3384419.3430898.

[119] Hu, Diyi und Krishnamachari, Bhaskar: „Fast and Accurate Streaming CNN Inference via Communication Compression on the Edge“. In: *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)*. 2020, S. 157–163. DOI: 10.1109/IoTDI49375.2020.00023.

[120] Eshratifar, Amir Erfan; Esmaili, Amirhossein und Pedram, Massoud: „BottleNet: A Deep Learning Architecture for Intelligent Mobile Cloud Computing Services“. In: *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. 2019, S. 1–6. DOI: 10.1109/ISLPED.2019.8824955.

[121] Shao, Jiawei und Zhang, Jun: „Communication-Computation Trade-off in Resource-Constrained Edge Inference“. In: *IEEE Communications Magazine* 58.12 (2020), S. 20–26. doi: 10.1109/MCOM.001.2000373.

[122] Dagli, Ismet; Cieslewicz, Alexander; McClurg, Jedidiah und Belviranli, Mehmet E.: „AxoNN: Energy-Aware Execution of Neural Network Inference on Multi-Accelerator Heterogeneous SoCs“. In: *Proceedings of the 59th ACM/IEEE Design Automation Conference*. DAC '22. San Francisco, California: Association for Computing Machinery, 2022, S. 1069–1074. doi: 10.1145/3489517.3530572. url: https://doi.org/10.1145/3489517.3530572.

[123] Van Delm, Josse; Vandersteegen, Maarten; Burrello, Alessio; Sarda, Giuseppe Maria; Conti, Francesco; Pagliari, Daniele Jahier; Benini, Luca und Verhelst, Marian: „HTVM: Efficient Neural Network Deployment On Heterogeneous TinyML Platforms“. In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 2023, S. 1–6. doi: 10.1109/DAC56929.2023.10247664.

[124] LiKamWa, Robert; Hou, Yunhui; Gao, Julian; Polansky, Mia und Zhong, Lin: „RedEye: analog ConvNet image sensor architecture for continuous mobile vision“. In: *SIGARCH Comput. Archit. News* 44.3 (Juni 2016), S. 255–266. doi: 10.1145/3007787.3001164. url: https://doi.org/10.1145/3007787.3001164.

[125] Ko, Jong Hwan; Na, Taesik; Amir, Mohammad Faisal und Mukhopadhyay, Saibal: „Edge-Host Partitioning of Deep Neural Networks with Feature Space Encoding for Resource-Constrained Internet-of-Things Platforms“. In: *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. 2018, S. 1–6. doi: 10.1109/AVSS.2018.8639121.

[126] Wen, Tai-Hao; Hung, Je-Min; Huang, Wei-Hsing; Jhang, Chuan-Jia; Lo, Yun-Chen; Hsu, Hung-Hsi; Ke, Zhao-En; Chen, Yu-Chiao; Chin, Yu-Hsiang; Su, Chin-I u. a.: „Fusion of memristor and digital compute-in-memory processing for energy-efficient edge computing“. In: *Science* 384.6693 (2024), S. 325–332. doi: 10.1126/science.adf5538.

[127] White, Colin; Safari, Mahmoud; Sukthanker, Rhea; Ru, Binxin; Elsken, Thomas; Zela, Arber; Dey, Debadeepta und Hutter, Frank: Neural Architecture Search: Insights from 1000 Papers. 2023. arXiv: 2301.08727 [cs.LG]. url: https://arxiv.org/abs/2301.08727.

[128] Gajski, Daniel D. und Kuhn, Robert H.: „Guest Editors' Introduction: New VLSI Tools“. In: *Computer* 16.12 (1983), S. 11–14. doi: 10.1109/MC.1983.1654264.

[129] Paszke, Adam u. a.: „PyTorch: An Imperative Style, High-Performance Deep Learning Library“. In: *Advances in Neural Information Processing Systems 32*. Hrsg. von Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E. und Garnett, R. Curran Associates, Inc., 2019, S. 8024–8035. url: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[130] NVIDIA Corporation: The NVIDIA Deep Learning Accelerator. Nov. 2018. url: http://nvdla.org/ (besucht am 19. 07. 2024).

[131] Lee, Yi-Che; Hsu, Ting-Shuo; Chen, Chun-Tse; Liou, Jing-Jia und Lu, Juin-Ming: „NNSim: A Fast and Accurate SystemC/TLM Simulator for Deep Convo-

lutional Neural Network Accelerators“. In: *2019 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*. 2019, S. 1–4. DOI: 10.1109/VLSI-DAT.2019.8741950.

[132] Kim, Sunwoo; Wang, Jooho; Seo, Youngho; Lee, Sanghun; Park, Yeji; Park, Sungkyung und Park, Chester Sungchung: Transaction-level Model Simulator for Communication-Limited Accelerators. 2020. DOI: 10.48550/ARXIV.2007.14897. URL: https://arxiv.org/abs/2007.14897.

[133] Abdelfattah, M. S.; Dudziak, Ł.; Chau, T.; Lee, R.; Kim, H. und Lane, N. D.: „Best of Both Worlds: AutoML Codesign of a CNN and its Hardware Accelerator“. In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 2020, S. 1–6. DOI: 10.1109/DAC18072.2020.9218596.

[134] Muñoz-Martínez, F.; Abellán, José L.; Acacio, M. und Krishna, T.: „STONNE: A Detailed Architectural Simulator for Flexible Neural Network Accelerators“. In: *ArXiv* abs/2006.07137 (2020).

[135] Samajdar, Ananda; Zhu, Yuhao; Whatmough, Paul; Mattina, Matthew und Krishna, Tushar: „SCALE-Sim: Systolic CNN Accelerator Simulator“. In: *arXiv preprint arXiv:1811.02883* (2018).

[136] Xi, Sam (Likun); Yao, Yuan; Bhardwaj, Kshitij; Whatmough, Paul; Wei, Gu-Yeon und Brooks, David: „SMAUG: End-to-End Full-Stack Simulation Infrastructure for Deep Learning Workloads“. In: *ACM Trans. Archit. Code Optim.* 17.4 (Nov. 2020). DOI: 10.1145/3424669. URL: https://doi.org/10.1145/3424669.

[137] Mei, Linyan; Houshmand, Pouya; Jain, Vikram; Giraldo, Sebastian und Verhelst, Marian: „ZigZag: Enlarging Joint Architecture-Mapping Design Space Exploration for DNN Accelerators“. In: *IEEE Transactions on Computers* 70.8 (2021), S. 1160–1174. DOI: 10.1109/TC.2021.3059962.

[138] Yang, Xuan u. a.: „Interstellar: Using Halide's Scheduling Language to Analyze DNN Accelerators“. In: *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '20. Lausanne, Switzerland: Association for Computing Machinery, 2020, S. 369–383. DOI: 10.1145/3373376.3378514. URL: https://doi.org/10.1145/3373376.3378514.

[139] Parashar, Angshuman; Raina, Priyanka; Shao, Yakun Sophia; Chen, Yu-Hsin; Ying, Victor A.; Mukkara, Anurag; Venkatesan, Rangharajan; Khailany, Brucek; Keckler, Stephen W. und Emer, Joel: „Timeloop: A Systematic Approach to DNN Accelerator Evaluation“. In: *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2019, S. 304–315. DOI: 10.1109/ISPASS.2019.00042.

[140] Guo, Kaiyuan; Zeng, Shulin; Yu, Jincheng; Wang, Yu und Yang, Huazhong: A Survey of FPGA-Based Neural Network Accelerator. 2018. arXiv: 1712.08934 [cs.AR].

[141] Wu, Yannan Nellie; Emer, Joel S. und Sze, Vivienne: „Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs“. In: *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2019, S. 1–8. DOI: 10.1109/ICCAD45719.2019.8942149.

[142] Li, S.; Chen, K.; Ahn, J. H.; Brockman, J. B. und Jouppi, N. P.: „CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques“. In: *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2011, S. 694–701. doi: 10.1109/ICCAD.2011.6105405.

[143] Shao, Y. S.; Reagen, B.; Wei, G. und Brooks, D.: „Aladdin: A pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures“. In: *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. 2014, S. 97–108. doi: 10.1109/ISCA.2014.6853196.

[144] Wilton, S.J.E. und Jouppi, N.P.: „CACTI: an enhanced cache access and cycle time model“. In: *IEEE Journal of Solid-State Circuits* 31.5 (1996), S. 677–688. doi: 10.1109/4.509850.

[145] Balasubramonian, Rajeev; Kahng, Andrew B.; Muralimanohar, Naveen; Shafiee, Ali und Srinivas, Vaishnav: „CACTI 7: New Tools for Interconnect Exploration in Innovative Off-Chip Memories“. In: *ACM Trans. Archit. Code Optim.* 14.2 (Juni 2017). doi: 10.1145/3085572. url: https://doi.org/10.1145/3085572.

[146] Ravipati, Divya Praneetha; Kedia, Rajesh; Van Santen, Victor M.; Henkel, Jörg; Panda, Preeti Ranjan und Amrouch, Hussam: „FN-CACTI: Advanced CACTI for FinFET and NC-FinFET Technologies“. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 30.3 (2022), S. 339–352. doi: 10.1109/TVLSI.2021.3123112.

[147] Reviriego, P.; Maestro, J.A.; Hernández, J.A. und Larrabeiti, D.: „Study of the potential energy savings in Ethernet by combining Energy Efficient Ethernet and Adaptive Link Rate“. In: *Transactions on Emerging Telecommunications Technologies* 23.3 (2012), S. 227–233. doi: https://doi.org/10.1002/ett.1526. url: https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.1526.

[148] Gámiz-Caro, Juan und Grau, Antoni: „MESSAGE DELAY IN DISTRIBUTED CONTROL SYSTEMS THROUGH ETHERNET“. In: *IFAC Proceedings Volumes* 38.1 (2005). 16th IFAC World Congress, S. 36–41. doi: https://doi.org/10.3182/20050703-6-CZ-1902.01161. url: https://www.sciencedirect.com/science/article/pii/S1474667016371737.

[149] NVIDIA Corporation: NVIDIA Jetson TX2. url: https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/jetson-tx2/ (besucht am 23. 10. 2024).

[150] Shelhamer, Evan; Long, Jonathan und Darrell, Trevor: „Fully Convolutional Networks for Semantic Segmentation“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.4 (2017), S. 640–651. doi: 10.1109/TPAMI.2016.2572683.

[151] Szegedy, Christian; Liu, Wei; Jia, Yangqing; Sermanet, Pierre; Reed, Scott; Anguelov, Dragomir; Erhan, Dumitru; Vanhoucke, Vincent und Rabinovich, Andrew: „Going deeper with convolutions“. In: *2015 IEEE Conference on*

Computer Vision and Pattern Recognition (CVPR). 2015, S. 1–9. DOI: 10.1109/CVPR.2015.7298594.

[152] IANDOLA, Forrest N.; HAN, Song; MOSKEWICZ, Matthew W.; ASHRAF, Khalid; DALLY, William J. und KEUTZER, Kurt: „SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size". In: *arXiv:1602.07360 [cs]* (Nov. 2016).

[153] PAPPALARDO, Alessandro: Xilinx/brevitas. 2021. DOI: 10.5281/zenodo.3333552. URL: https://doi.org/10.5281/zenodo.3333552.

[154] BLOTT, Michaela; PREUßER, Thomas B.; FRASER, Nicholas J.; GAMBARDELLA, Giulio; O'BRIEN, Kenneth; UMUROGLU, Yaman; LEESER, Miriam und VISSERS, Kees: „FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks". In: *ACM Trans. Reconfigurable Technol. Syst.* 11.3 (Dez. 2018). DOI: 10.1145/3242897. URL: https://doi.org/10.1145/3242897.

[155] HOWARD, Andrew G.; ZHU, Menglong; CHEN, Bo; KALENICHENKO, Dmitry; WANG, Weijun; WEYAND, Tobias; ANDREETTO, Marco und ADAM, Hartwig: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. 2017. DOI: 10.48550/ARXIV.1704.04861. URL: https://arxiv.org/abs/1704.04861.

[156] CORMEN, Thomas H; LEISERSON, Charles E; RIVEST, Ronald L und STEIN, Clifford: Introduction to algorithms. MIT press, 2022.

[157] LINUX FOUNDATION: ONNX: Open Neural Network Exchange. URL: https://github.com/onnx/onnx (besucht am 19. 07. 2024).

[158] HAGBERG, Aric; SWART, Pieter J. und SCHULT, Daniel A.: „Exploring network structure, dynamics, and function using NetworkX". In: (Jan. 2008). URL: https://www.osti.gov/biblio/960616.

[159] PETRA, Michel; ULRICH, Lauther und PETER, Duzy: The Synthesis Approach to Digital System Design. The Kluwer International Series in Engineering and Computer Science. VLSI, Computer Architecture, and Digital Signal Processing VLSI, computer architecture, and digital signal processing. Springer, 2012.

[160] KAHN, A. B.: „Topological sorting of large networks". In: *Commun. ACM* 5.11 (Nov. 1962), S. 558–562. DOI: 10.1145/368996.369025. URL: https://doi.org/10.1145/368996.369025.

[161] ZHU, Zhenhua u. a.: „MNSIM 2.0: A Behavior-Level Modeling Tool for Memristor-based Neuromorphic Computing Systems". In: *Proceedings of the 2020 on Great Lakes Symposium on VLSI*. GLSVLSI '20. Virtual Event, China: Association for Computing Machinery, 2020, S. 83–88. DOI: 10.1145/3386263.3407647. URL: https://doi.org/10.1145/3386263.3407647.

[162] SUN, Hanbo; ZHU, Zhenhua; WANG, Chenyu; NING, Xuefei; DAI, Guohao; YANG, Huazhong und WANG, Yu: „Gibbon: An efficient co-exploration framework of nn model and processing-in-memory architecture". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2023).

[163] VIRTANEN, Pauli u. a.: „SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), S. 261–272. DOI: 10.1038/s41592-019-0686-2.

[164] Zangeneh, Mahmoud und Joshi, Ajay: „Design and Optimization of Nonvolatile Multibit 1T1R Resistive RAM“. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22.8 (2014), S. 1815–1828. doi: 10.1109/TVLSI.2013.2277715.

[165] Zhu, Zhenhua; Sun, Hanbo; Lin, Yujun; Dai, Guohao; Xia, Lixue; Han, Song; Wang, Yu und Yang, Huazhong: „A configurable multi-precision CNN computing framework based on single bit RRAM“. In: *Proceedings of the 56th Annual Design Automation Conference 2019*. 2019, S. 1–6.

[166] Cai, Yi; Tang, Tianqi; Xia, Lixue; Li, Boxun; Wang, Yu und Yang, Huazhong: „Low Bit-Width Convolutional Neural Network on RRAM“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.7 (2020), S. 1414–1427. doi: 10.1109/TCAD.2019.2917852.

[167] Haensch, Wilfried; Raghunathan, Anand; Roy, Kaushik; Chakrabarti, Bhaswar; Phatak, Charudatta M.; Wang, Cheng und Guha, Supratik: „Compute in-Memory with Non-Volatile Elements for Neural Networks: A Review from a Co-Design Perspective“. In: *Advanced Materials* 35.37 (2023). doi: https://doi.org/10.1002/adma.202204944. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/adma.202204944. url: https://onlinelibrary.wiley.com/doi/abs/10.1002/adma.202204944.

[168] Banerjee, Writam: „Challenges and Applications of Emerging Nonvolatile Memory Devices“. In: *Electronics* 9.6 (2020). doi: 10.3390/electronics9061029. url: https://www.mdpi.com/2079-9292/9/6/1029.

[169] Mannocci, P.; Farronato, M.; Lepri, N.; Cattaneo, L.; Glukhov, A.; Sun, Z. und Ielmini, D.: „In-memory computing with emerging memory devices: Status and outlook“. In: *APL Machine Learning* 1.1 (Feb. 2023). doi: 10.1063/5.0136403. eprint: https://pubs.aip.org/aip/aml/article-pdf/doi/10.1063/5.0136403/19999534/010902_1_5.0136403.pdf. url: https://doi.org/10.1063/5.0136403.

[170] Seol, Youhwan; Hyeon, Doyeon; Min, Junhong; Kim, Moonbeom und Paek, Jeongyeup: „Timely Survey of Time-Sensitive Networking: Past and Future Directions“. In: *IEEE Access* 9 (2021), S. 142506–142527. doi: 10.1109/ACCESS.2021.3120769.

[171] Kim, Haeri; Yoo, Wonsuk; Ha, Seoncheol und Chung, Jong-Moon: „In-Vehicle Network Average Response Time Analysis for CAN-FD and Automotive Ethernet“. In: *IEEE Transactions on Vehicular Technology* 72.6 (2023), S. 6916–6932. doi: 10.1109/TVT.2023.3236593.

[172] Lo Bello, Lucia; Patti, Gaetano und Leonardi, Luca: „A Perspective on Ethernet in Automotive Communications—Current Status and Future Trends“. In: *Applied Sciences* 13.3 (2023). doi: 10.3390/app13031278. url: https://www.mdpi.com/2076-3417/13/3/1278.

[173] De Vincenzi, Marco; Costantino, Gianpiero; Matteucci, Ilaria; Fenzl, Florian; Plappert, Christian; Rieke, Roland und Zelle, Daniel: „A Systematic Review on Security Attacks and Countermeasures in Automotive Ethernet“.

In: *ACM Comput. Surv.* 56.6 (Jan. 2024). DOI: 10.1145/3637059. URL: https://doi.org/10.1145/3637059.

[174] Kannan, Ajaykumar; Jerger, Natalie Enright und Loh, Gabriel H.: „Enabling interposer-based disintegration of multi-core processors“. In: *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2015, S. 546–558. DOI: 10.1145/2830772.2830808.

[175] Vivet, Pascal u. a.: „IntAct: A 96-Core Processor With Six Chiplets 3D-Stacked on an Active Interposer With Distributed Interconnects and Integrated Power Management“. In: *IEEE Journal of Solid-State Circuits* 56.1 (2021), S. 79–97. DOI: 10.1109/JSSC.2020.3036341.

[176] Kim, Jinwoo u. a.: „Architecture, Chip, and Package Co-design Flow for 2.5D IC Design Enabling Heterogeneous IP Reuse“. In: *Proceedings of the 56th Annual Design Automation Conference 2019*. DAC '19. Las Vegas, NV, USA: Association for Computing Machinery, 2019. DOI: 10.1145/3316781.3317775. URL: https://doi.org/10.1145/3316781.3317775.

[177] Das Sharma, Debendra; Pasdast, Gerald; Qian, Zhiguo und Aygun, Kemal: „Universal Chiplet Interconnect Express (UCIe): An Open Industry Standard for Innovations With Chiplets at Package Level“. In: *IEEE Transactions on Components, Packaging and Manufacturing Technology* 12.9 (2022), S. 1423–1431. DOI: 10.1109/TCPMT.2022.3207195.

[178] Sharma, Harsh; Mandal, Sumit K.; Doppa, Janardhan Rao; Ogras, Umit und Pande, Partha Pratim: „Achieving Datacenter-scale Performance through Chiplet-based Manycore Architectures“. In: *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2023, S. 1–6. DOI: 10.23919/DATE56975.2023.10137125.

[179] Hayes, Conor F. u. a.: „A practical guide to multi-objective reinforcement learning and planning“. In: *Autonomous Agents and Multi-Agent Systems* 36.1 (Apr. 2022), S. 26. DOI: 10.1007/s10458-022-09552-y. URL: https://doi.org/10.1007/s10458-022-09552-y.

[180] Blank, J. und Deb, K.: „pymoo: Multi-Objective Optimization in Python“. In: *IEEE Access* 8 (2020), S. 89497–89509.

[181] Xia, Lixue; Huangfu, Wenqin; Tang, Tianqi; Yin, Xiling; Chakrabarty, Krishnendu; Xie, Yuan; Wang, Yu und Yang, Huazhong: „Stuck-at Fault Tolerance in RRAM Computing Systems“. In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 8.1 (2018), S. 102–115. DOI: 10.1109/JETCAS.2017.2776980.

[182] Chen, Ching-Yi; Shih, Hsiu-Chuan; Wu, Cheng-Wen; Lin, Chih-He; Chiu, Pi-Feng; Sheu, Shyh-Shyuan und Chen, Frederick T.: „RRAM Defect Modeling and Failure Analysis Based on March Test and a Novel Squeeze-Search Scheme“. In: *IEEE Transactions on Computers* 64.1 (2015), S. 180–190. DOI: 10.1109/TC.2014.12.

[183] Lee, Matthew Kay Fei; Cui, Yingnan; Somu, Thannirmalai; Luo, Tao; Zhou, Jun; Tang, Wai Teng; Wong, Weng-Fai und Goh, Rick Siow Mong: „A System-Level Simulator for RRAM-Based Neuromorphic Computing Chips“. In: *ACM*

Trans. Archit. Code Optim. 15.4 (Jan. 2019). DOI: 10.1145/3291054. URL: https://doi.org/10.1145/3291054.

[184] Xia, Lixue; Liu, Mengyun; Ning, Xuefei; Chakrabarty, Krishnendu und Wang, Yu: „Fault-Tolerant Training Enabled by On-Line Fault Detection for RRAM-Based Neural Computing Systems“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38.9 (2019), S. 1611–1624. DOI: 10.1109/TCAD.2018.2855145.

[185] Ning, Xuefei; Ge, Guangjun; Li, Wenshuo; Zhu, Zhenhua; Zheng, Yin; Chen, Xiaoming; Gao, Zhen; Wang, Yu und Yang, Huazhong: „FTT-NAS: Discovering Fault-tolerant Convolutional Neural Architecture“. In: *ACM Trans. Des. Autom. Electron. Syst.* 26.6 (Aug. 2021). DOI: 10.1145/3460288. URL: https://doi.org/10.1145/3460288.

[186] Zhang, Tianyun; Ye, Shaokai; Zhang, Kaiqi; Tang, Jian; Wen, Wujie; Fardad, Makan und Wang, Yanzhi: „A Systematic DNN Weight Pruning Framework using Alternating Direction Method of Multipliers“. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Sep. 2018.

[187] Liu, Zhuang; Sun, Mingjie; Zhou, Tinghui; Huang, Gao und Darrell, Trevor: Rethinking the Value of Network Pruning. 2019. arXiv: 1810.05270 [cs.LG]. URL: https://arxiv.org/abs/1810.05270.

[188] Ogawa, Tomoya u. a.: „15.8 A 22nm 10.8Mb Embedded STT-MRAM Macro Achieving over 200MHz Random-Read Access and a 10.4MB/s Write Throughput with an In-Field Programmable 0.3Mb MTJ-OTP for High-End MCUs“. In: *2024 IEEE International Solid-State Circuits Conference (ISSCC)*. Bd. 67. 2024, S. 290–292. DOI: 10.1109/ISSCC49657.2024.10454409.

[189] Wang, Qishen u. a.: „A Logic-Process Compatible RRAM with 15.43 Mb/mm2 Density and 10years@150°C retention using STI-less Dynamic-Gate and Self-Passivation Sidewall“. In: *2023 International Electron Devices Meeting (IEDM)*. 2023, S. 1–4. DOI: 10.1109/IEDM45741.2023.10413885.

[190] Swaileh, Wassim; Imbert, Florent; Soullard, Yann; Tavenard, Romain und Anquetil, Éric: „Online handwriting trajectory reconstruction from kinematic sensors using temporal convolutional network“. In: *International Journal on Document Analysis and Recognition (IJDAR)* (2023), S. 1–14.

[191] Jain, Vikram; Giraldo, Sebastian; Roose, Jaro De; Mei, Linyan; Boons, Bert und Verhelst, Marian: „TinyVers: A Tiny Versatile System-on-Chip With State-Retentive eMRAM for ML Inference at the Extreme Edge“. In: *IEEE Journal of Solid-State Circuits* 58.8 (2023), S. 2360–2371. DOI: 10.1109/JSSC.2023.3236566.

[192] Loh, Johnson und Gemmeke, Tobias: „Dataflow Optimizations in a Sub-uW Data-Driven TCN Accelerator for Continuous ECG Monitoring“. In: *2022 IEEE Nordic Circuits and Systems Conference (NorCAS)*. 2022, S. 1–7. DOI: 10.1109/NorCAS57515.2022.9934591.

[193] STABILO International GmbH: The STABILO Digipen. URL: https://stabilodigital.com/ (besucht am 13. 06. 2024).

[194] Tosi, Jacopo; Taffoni, Fabrizio; Santacatterina, Marco; Sannino, Roberto und Formica, Domenico: „Performance Evaluation of Bluetooth Low Energy:

A Systematic Review“. In: *Sensors* 17.12 (2017). DOI: 10.3390/s17122898. URL: https://www.mdpi.com/1424-8220/17/12/2898.

[195] Wehbi, Mohamad; Luge, Daniel; Hamann, Tim; Barth, Jens; Kaempf, Peter; Zanca, Dario und Eskofier, Bjoern M.: „Surface-Free Multi-Stroke Trajectory Reconstruction and Word Recognition Using an IMU-Enhanced Digital Pen“. In: *Sensors* 22.14 (2022). DOI: 10.3390/s22145347. URL: https://www.mdpi.com/1424-8220/22/14/5347.

[196] Conti, Francesco: Technical Report: NEMO DNN Quantization for Deployment Model. 2020. arXiv: 2004.05930.

[197] Graves, Alex; Fernández, Santiago; Gomez, Faustino und Schmidhuber, Jürgen: „Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks“. In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 2006, S. 369–376. DOI: 10.1145/1143844.1143891. URL: https://doi.org/10.1145/1143844.1143891.

[198] Wehbi, Mohamad; Hamann, Tim; Barth, Jens; Kaempf, Peter; Zanca, Dario und Eskofier, Bjoern: „Towards an IMU-based Pen Online Handwriting Recognizer“. In: *Document Analysis and Recognition – ICDAR 2021*. Hrsg. von Lladós, Josep; Lopresti, Daniel und Uchida, Seiichi. Cham: Springer International Publishing, 2021, S. 289–303.

[199] Scheidl, H.: CTC Decoding Algorithms. URL: https://github.com/githubharald/CTCDecoder (besucht am 19. 07. 2024).

[200] Levenshtein, Vladimir I u. a.: „Binary codes capable of correcting deletions, insertions, and reversals“. In: *Soviet physics doklady*. Bd. 10. 8. Soviet Union. 1966, S. 707–710.

[201] Jurafsky, D. und Martin, J. H.: „N-gram Language Models“. In: *Speech and Language Processing*. 2021.

[202] Imperas Software Limited: OVPsim. URL: https://ovpworld.org (besucht am 18. 06. 2024).

[203] RISC-V foundation: Spike RISC-V ISA Simulator. URL: https://github.com/riscv-software-src/riscv-isa-sim (besucht am 18. 06. 2024).

[204] Rossi, Davide u. a.: „Vega: A Ten-Core SoC for IoT Endnodes With DNN Acceleration and Cognitive Wake-Up From MRAM-Based State-Retentive Sleep Mode“. In: *IEEE Journal of Solid-State Circuits* 57.1 (2022), S. 127–139. DOI: 10.1109/JSSC.2021.3114881.

[205] Rybalkin, Vladimir; Sudarshan, Chirag; Weis, Christian; Lappas, Jan; Wehn, Norbert und Cheng, Li: „Efficient hardware architectures for 1D-and MD-LSTM networks“. In: *Journal of Signal Processing Systems* 92 (2020), S. 1219–1245.

[206] Meißl, Florian; Eibensteiner, Florian; Petz, Phillip und Langer, Josef: „Online Handwriting Recognition using LSTM on Microcontroller and IMU Sensors“. In: *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*. 2022, S. 999–1004. DOI: 10.1109/ICMLA55696.2022.00167.

[207] Conti, Francesco; Cavigelli, Lukas; Paulin, Gianna; Susmelj, Igor und Benini, Luca: „Chipmunk: A systolically scalable 0.9 mm2, 3.08Gop/s/mW @ 1.2 mW accelerator for near-sensor recurrent neural network inference“. In: *2018 IEEE Custom Integrated Circuits Conference (CICC)*. 2018, S. 1–4. DOI: 10.1109/CICC.2018.8357068.

[208] Azari, Elham und Vrudhula, Sarma: „An Energy-Efficient Reconfigurable LSTM Accelerator for Natural Language Processing“. In: *2019 IEEE International Conference on Big Data (Big Data)*. 2019, S. 4450–4459. DOI: 10.1109/BigData47090.2019.9006030.

[209] Chen, Kewei; Huang, Leilei; Li, Minjiang; Zeng, Xiaoyang und Fan, Yibo: „A Compact and Configurable Long Short-Term Memory Neural Network Hardware Architecture“. In: *2018 25th IEEE International Conference on Image Processing (ICIP)*. 2018, S. 4168–4172. DOI: 10.1109/ICIP.2018.8451053.

[210] Zhang, Weifeng; Ge, Fen; Cui, Chenchen; Yang, Ying; Zhou, Fang und Wu, Ning: „Design and Implementation of LSTM Accelerator Based on FPGA“. In: *2020 IEEE 20th International Conference on Communication Technology (ICCT)*. 2020, S. 1675–1679. DOI: 10.1109/ICCT50939.2020.9295665.

[211] Que, Zhiqiang; Nakahara, Hiroki; Nurvitadhi, Eriko; Boutros, Andrew; Fan, Hongxiang; Zeng, Chenglong; Meng, Jiuxi; Tsoi, Kuen Hung; Niu, Xinyu und Luk, Wayne: „Recurrent Neural Networks With Column-Wise Matrix-Vector Multiplication on FPGAs“. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 30.2 (2022), S. 227–237. DOI: 10.1109/TVLSI.2021.3135353.

[212] Bank-Tavakoli, Erfan; Ghasemzadeh, Seyed Abolfazl; Kamal, Mehdi; Afzali-Kusha, Ali und Pedram, Massoud: „POLAR: A Pipelined/Overlapped FPGA-Based LSTM Accelerator“. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28.3 (2020), S. 838–842. DOI: 10.1109/TVLSI.2019.2947639.

[213] Chen, Jeffrey; Hong, Sehwan; He, Warrick; Moon, Jinyeong und Jun, Sang-Woo: „Eciton: Very Low-Power LSTM Neural Network Accelerator for Predictive Maintenance at the Edge“. In: *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*. 2021, S. 1–8. DOI: 10.1109/FPL53798.2021.00009.

[214] Qian, Chao; Ling, Tianheng und Schiele, Gregor: „Enhancing Energy-Efficiency by Solving the Throughput Bottleneck of LSTM Cells for Embedded FPGAs“. In: *Machine Learning and Principles and Practice of Knowledge Discovery in Databases*. Hrsg. von Koprinska, Irena u. a. Cham: Springer Nature Switzerland, 2023, S. 594–605.

[215] Boutros, Andrew; Yazdanshenas, Sadegh und Betz, Vaughn: „Embracing Diversity: Enhanced DSP Blocks for Low-Precision Deep Learning on FPGAs“. In: *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. 2018, S. 35–357. DOI: 10.1109/FPL.2018.00014.

[216] Mitchell, John N.: „Computer Multiplication and Division Using Binary Logarithms“. In: *IRE Transactions on Electronic Computers* EC-11.4 (1962), S. 512–517. DOI: 10.1109/TEC.1962.5219391.

[217] Bulić, Patricio; Babić, Zdenka und Avramović, Aleksej: „A simple pipelined logarithmic multiplier“. In: *2010 IEEE International Conference on Computer Design*. 2010, S. 235–240. doi: 10.1109/ICCD.2010.5647767.

[218] Namin, Ashkan Hosseinzadeh; Leboeuf, Karl; Wu, Huapeng und Ahmadi, Majid: „Artificial neural networks activation function HDL coder“. In: *2009 IEEE International Conference on Electro/Information Technology*. 2009, S. 389–392. doi: 10.1109/EIT.2009.5189648.

[219] Kumar Meher, Pramod: „An optimized lookup-table for the evaluation of sigmoid function for artificial neural networks“. In: *2010 18th IEEE/IFIP International Conference on VLSI and System-on-Chip*. 2010, S. 91–95. doi: 10.1109/VLSISOC.2010.5642617.

[220] Coelho, Claudionor N.; Kuusela, Aki; Li, Shan; Zhuang, Hao; Ngadiuba, Jennifer; Aarrestad, Thea Klaeboe; Loncar, Vladimir; Pierini, Maurizio; Pol, Adrian Alan und Summers, Sioni: „Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors“. In: *Nature Machine Intelligence* 3.8 (Juni 2021), S. 675–686. doi: 10.1038/s42256-021-00356-5. url: https://doi.org/10.1038/s42256-021-00356-5.

[221] Nolting, Stephan u. a.: The NEORV32 RISC-V Processor. 2023. doi: 10.5281/zenodo.5018888. url: https://doi.org/10.5281/zenodo.5018888.

[222] Kingma, Diederik P. und Ba, Jimmy: Adam: A Method for Stochastic Optimization. 2017. arXiv: 1412.6980.

[223] Bai, Shaojie; Kolter, J. Zico und Koltun, Vladlen: An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. 2018. arXiv: 1803.01271 [cs.LG].

[224] Ingolfsson, Thorir Mar u. a.: „ECG-TCN: Wearable Cardiac Arrhythmia Detection with a Temporal Convolutional Network“. In: *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. 2021, S. 1–4. doi: 10.1109/AICAS51828.2021.9458520.

[225] Langer, Patrick; Haddadi Esfahani, Ali; Dyka, Zoya und Langendörfer, Peter: „FPGA-Based Realtime Detection of Freezing of Gait of Parkinson Patients“. In: *Body Area Networks. Smart IoT and Big Data for Intelligent Health Management*. Hrsg. von Ur Rehman, Masood und Zoha, Ahmed. Cham: Springer International Publishing, 2022, S. 101–111.

[226] Carreras, Marco; Deriu, Gianfranco; Raffo, Luigi; Benini, Luca und Meloni, Paolo: „Optimizing Temporal Convolutional Network Inference on FPGA-Based Accelerators“. In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 10.3 (2020), S. 348–361. doi: 10.1109/JETCAS.2020.3014503.

[227] Bernardo, Paul Palomero; Gerum, Christoph; Frischknecht, Adrian; Lübeck, Konstantin und Bringmann, Oliver: „UltraTrail: A Configurable Ultralow-Power TC-ResNet AI Accelerator for Efficient Keyword Spotting“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.11 (2020), S. 4240–4251. doi: 10.1109/TCAD.2020.3012320.

[228] Giraldo, J. S. P.; Jain, Vikram und Verhelst, Marian: „Efficient Execution of Temporal Convolutional Networks for Embedded Keyword Spotting“. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 29.12 (2021), S. 2220–2228. doi: 10.1109/TVLSI.2021.3120189.

[229] Khatwani, Mohit u. a.: „Energy Efficient Convolutional Neural Networks for EEG Artifact Detection“. In: *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. 2018, S. 1–4. doi: 10.1109/BIOCAS.2018.8584791.

[230] Flamand, Eric u. a.: „GAP-8: A RISC-V SoC for AI at the Edge of the IoT“. In: *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. 2018, S. 1–4. doi: 10.1109/ASAP.2018.8445101.

[231] Akiba, Takuya; Sano, Shotaro; Yanase, Toshihiko; Ohta, Takeru und Koyama, Masanori: „Optuna: A Next-generation Hyperparameter Optimization Framework“. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[232] Saha, Swapnil Sayan; Sandha, Sandeep Singh; Garcia, Luis Antonio und Srivastava, Mani: „TinyOdom: Hardware-Aware Efficient Neural Inertial Navigation“. In: *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 6.2 (Juli 2022). doi: 10.1145/3534594. url: https://doi.org/10.1145/3534594.

[233] Herath, Sachini; Yan, Hang und Furukawa, Yasutaka: „RoNIN: Robust Neural Inertial Navigation in the Wild: Benchmark, Evaluations, & New Methods“. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, S. 3146–3152. doi: 10.1109/ICRA40945.2020.9196860.

[234] Warden, P.: „Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition“. In: *ArXiv e-prints* (Apr. 2018). arXiv: 1804.03209 [cs.CL]. url: https://arxiv.org/abs/1804.03209.

[235] Hirsch, Hans-Günter; Hellwig, K und Dobler, Stefan: „Speech recognition at multiple sampling rates.“ In: *INTERSPEECH*. 2001, S. 1837–1840.

[236] Gizopoulos, Dimitris; Psarakis, Mihalis; Adve, Sarita V.; Ramachandran, Pradeep; Hari, Siva Kumar Sastry; Sorin, Daniel; Meixner, Albert; Biswas, Arijit und Vera, Xavier: „Architectures for online error detection and recovery in multicore processors“. In: *2011 Design, Automation & Test in Europe*. 2011, S. 1–6. doi: 10.1109/DATE.2011.5763096.

[237] Koo, R. und Toueg, S.: „Checkpointing and Rollback-Recovery for Distributed Systems“. In: *IEEE Transactions on Software Engineering* SE-13.1 (1987), S. 23–31. doi: 10.1109/TSE.1987.232562.

[238] Teodorescu, R.; Nakano, Jun und Torrellas, J.: „SWICH: A Prototype for Efficient Cache-Level Checkpointing and Rollback“. In: *IEEE Micro* 26.5 (2006), S. 28–40. doi: 10.1109/MM.2006.100.

[239] Nakada, Takashi und Nakamura, Hiroshi: „Normally-Off Computing“. In: *Normally-Off Computing*. Hrsg. von Nakada, Takashi und Nakamura, Hiroshi. Tokyo: Springer Japan, 2017, S. 57–63. doi: 10.1007/978-4-431-56505-5_4.

[240] Bishnoi, Rajendra; Oboril, Fabian und Tahoori, Mehdi B.: „Non-Volatile Non-Shadow flip-flop using Spin Orbit Torque for efficient normally-off com-

puting“. In: *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2016, S. 769–774. DOI: 10.1109/ASPDAC.2016.7428104.

[241] Kvatinsky, Shahar; Nacson, Yuval H.; Etsion, Yoav; Friedman, Eby G.; Kolodny, Avinoam und Weiser, Uri C.: „Memristor-Based Multithreading“. In: *IEEE Computer Architecture Letters* 13.1 (2014), S. 41–44. DOI: 10.1109/L-CA.2013.3.

[242] Wang, Yiqun; Liu, Yongpan; Liu, Yumeng; Zhang, Daming; Li, Shuangchen; Sai, Baiko; Chiang, Mei-Fang und Yang, Huazhong: „A compression-based area-efficient recovery architecture for nonvolatile processors“. In: *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2012, S. 1519–1524. DOI: 10.1109/DATE.2012.6176714.

[243] Sun, Guangyu; Zhao, Jishen; Poremba, Matt; Xu, Cong und Xie, Yuan: „Memory that never forgets: emerging nonvolatile memory and the implication for architecture design“. In: *National Science Review* 5.4 (Aug. 2017), S. 577–592. DOI: 10.1093/nsr/nwx082. eprint: https://academic.oup.com/nsr/article-pdf/5/4/577/31567686/nwx082.pdf. URL: https://doi.org/10.1093/nsr/nwx082.

[244] Chakraborty, Indranil; Jaiswal, A; Saha, AK; Gupta, SK und Roy, K: „Pathways to efficient neuromorphic computing with non-volatile memory technologies“. In: *Applied Physics Reviews* 7.2 (2020). DOI: 10.1063/1.5113536.

[245] Endoh, Tetsuo; Koike, Hiroki; Ikeda, Shoji; Hanyu, Takahiro und Ohno, Hideo: „An Overview of Nonvolatile Emerging Memories— Spintronics for Working Memories“. In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 6.2 (2016), S. 109–119. DOI: 10.1109/JETCAS.2016.2547704.

[246] Apalkov, Dmytro; Dieny, Bernard und Slaughter, J. M.: „Magnetoresistive Random Access Memory“. In: *Proceedings of the IEEE* 104.10 (2016), S. 1796–1830. DOI: 10.1109/JPROC.2016.2590142.

[247] Kim, Jongyeon; Chen, An; Behin-Aein, Behtash; Kumar, Saurabh; Wang, Jian-Ping und Kim, Chris H.: „A technology-agnostic MTJ SPICE model with user-defined dimensions for STT-MRAM scalability studies“. In: *2015 IEEE Custom Integrated Circuits Conference (CICC)*. 2015, S. 1–4. DOI: 10.1109/CICC.2015.7338407.

[248] Rossi, Davide u. a.: „Vega: A Ten-Core SoC for IoT Endnodes With DNN Acceleration and Cognitive Wake-Up From MRAM-Based State-Retentive Sleep Mode“. In: *IEEE Journal of Solid-State Circuits* 57.1 (2022), S. 127–139. DOI: 10.1109/JSSC.2021.3114881.

[249] Li, Tuo; Ragel, Roshan und Parameswaran, Sri: „Reli: Hardware/software Checkpoint and Recovery scheme for embedded processors“. In: *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2012, S. 875–880. DOI: 10.1109/DATE.2012.6176621.

[250] Ansari, Mohsen; Safari, Sepideh; Khdr, Heba; Gohari-Nazari, Pourya; Henkel, Jörg; Ejlali, Alireza und Hessabi, Shaahin: „Power-Aware Checkpointing for Multicore Embedded Systems“. In: *IEEE Transactions on Parallel and Distributed Systems* 33.12 (2022), S. 4410–4424. DOI: 10.1109/TPDS.2022.3188568.

[251] Li, Tuo; Ambrose, Jude Angelo und Parameswaran, Sri: „RECORD: Reducing register traffic for checkpointing in embedded processors“. In: *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2016, S. 582–587.

[252] Sheng, Xiao; Wang, Yiqun; Liu, Yongpan und Yang, Huazhong: „SPaC: A segment-based parallel compression for backup acceleration in nonvolatile processors“. In: *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2013, S. 865–868. doi: 10.7873/DATE.2013.182.

[253] Wang, Yiqun; Liu, Yongpan; Li, Shuangchen; Sheng, Xiao; Zhang, Daming; Chiang, Mei-Fang; Sai, Baiko; Hu, Xiaobo Sharon und Yang, Huazhong: „PaCC: A Parallel Compare and Compress Codec for Area Reduction in Nonvolatile Processors“. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22.7 (2014), S. 1491–1505. doi: 10.1109/TVLSI.2013.2275740.

[254] Senni, Sophiane; Torres, Lionel; Benoit, Pascal; Gamatie, Abdoulaye und Sassatelli, Gilles: „Normally-Off Computing and Checkpoint/Rollback for Fast, Low-Power, and Reliable Devices“. In: *IEEE Magnetics Letters* 8 (2017), S. 1–5. doi: 10.1109/LMAG.2017.2712780.

[255] Gebregiorgis, Anteneh; Bishnoi, Rajendra und Tahoori, Mehdi B.: „Spintronic normally-off heterogeneous system-on-chip design“. In: *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2018, S. 113–118. doi: 10.23919/DATE.2018.8341989.

[256] Chabi, Djaafar; Zhao, Weisheng; Deng, Erya; Zhang, Yue; Romdhane, Nesrine Ben; Klein, Jacques-Olivier und Chappert, Claude: „Ultra Low Power Magnetic Flip-Flop Based on Checkpointing/Power Gating and Self-Enable Mechanisms“. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 61.6 (2014), S. 1755–1765. doi: 10.1109/TCSI.2013.2295026.

[257] Darbari, Ashish; Al Hashimi, Bashir M.; Flynn, David und Biggs, John: „Selective state retention design using symbolic simulation“. In: *2009 Design, Automation & Test in Europe Conference & Exhibition*. 2009, S. 1644–1649. doi: 10.1109/DATE.2009.5090927.

[258] Baehr, Johanna; Hepp, Alexander; Brunner, Michaela; Malenko, Maja und Sigl, Georg: „Open Source Hardware Design and Hardware Reverse Engineering: A Security Analysis“. In: *2022 25th Euromicro Conference on Digital System Design (DSD)*. 2022, S. 504–512. doi: 10.1109/DSD57027.2022.00073.

[259] Brayton, Robert und Mishchenko, Alan: „ABC: An Academic Industrial-Strength Verification Tool“. In: *Computer Aided Verification*. Hrsg. von Touili, Tayssir; Cook, Byron und Jackson, Paul. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, S. 24–40.

[260] Wolf, Clifford; Glaser, Johann und Kepler, Johannes: „Yosys-a free Verilog synthesis suite“. In: *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*. 2013.

[261] Subramanyan, Pramod; Tsiskaridze, Nestan; Li, Wenchao; Gascón, Adrià; Tan, Wei Yang; Tiwari, Ashish; Shankar, Natarajan; Seshia, Sanjit A. und Malik, Sharad: „Reverse Engineering Digital Circuits Using Structural and

Functional Analyses“. In: *IEEE Transactions on Emerging Topics in Computing* 2.1 (2014), S. 63–80. DOI: 10.1109/TETC.2013.2294918.

[262] Rokicki, Simon; Pala, Davide; Paturel, Joseph und Sentieys, Olivier: „What You Simulate Is What You Synthesize: Designing a Processor Core from C++ Specifications“. In: *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2019, S. 1–8. DOI: 10.1109/ICCAD45719.2019.8942177.

[263] Siemens EDA: Catapult Ultra Synthesis 10.6a. 2021. URL: https://eda.sw.siemens.com/en-US/ic/catapult-high-level-synthesis/.

[264] Wolf, Clifford: PicoRV32 - A Size-Optimized RISC-V CPU. 2019. URL: https://github.com/YosysHQ/picorv32.

[265] liangkangnan: tinyriscv. 2020. URL: https://github.com/liangkangnan/tinyriscv.

Publikationen

[Kre22a] Kreß, Fabian; Hoefer, Julian; Hotfilter, Tim; Walter, Iris; Sidorenko, Vladimir; Harbaum, Tanja und Becker, Jürgen: „Hardware-aware Partitioning of Convolutional Neural Network Inference for Embedded AI Applications“. In: *2022 18th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. 2022, S. 133–140. doi: 10.1109/DCOSS54816.2022.00034.

[Kre22b] Kreß, Fabian; Serdyuk, Alexey; Hotfilter, Tim; Hoefer, Julian; Harbaum, Tanja; Becker, Jürgen und Hamann, Tim: „Hardware-aware Workload Distribution for AI-based Online Handwriting Recognition in a Sensor Pen“. In: *2022 11th Mediterranean Conference on Embedded Computing (MECO)*. 2022, S. 1–4. doi: 10.1109/MECO55406.2022.9797131.

[Kre23a] Kreß, Fabian; Hoefer, Julian; Hotfilter, Tim; Walter, Iris; El Annabi, El Mahdi; Harbaum, Tanja und Becker, Jürgen: „Automated Search for Deep Neural Network Inference Partitioning on Embedded FPGA“. In: *Machine Learning and Principles and Practice of Knowledge Discovery in Databases*. Hrsg. von Koprinska, Irena u. a. Cham: Springer Nature Switzerland, 2023, S. 557–568.

[Kre23b] Kreß, Fabian; Sidorenko, Vladimir; Schmidt, Patrick; Hoefer, Julian; Hotfilter, Tim; Walter, Iris; Harbaum, Tanja und Becker, Jürgen: „CNNParted: An open source framework for efficient Convolutional Neural Network inference partitioning in embedded systems“. In: *Computer Networks* 229 (2023), S. 109759. doi: https://doi.org/10.1016/j.comnet.2023.109759.

[Kre23c] Kreß, Fabian; Serdyuk, Alexey; Hiegle, Micha; Waldmann, Disnebio; Hotfilter, Tim; Hoefer, Julian; Hamann, Tim; Barth, Jens; Kämpf, Peter; Harbaum, Tanja und Becker, Juergen: „ATLAS: An Approximate Time-Series LSTM Accelerator for Low-Power IoT Applications“. In: *2023 26th Euromicro Conference on Digital System Design (DSD)*. 2023.

[Kre23d] Kreß, Fabian; Pfau, Johannes; Kempf, Fabian; Schmidt, Patrick; He, Zhuofan; Harbaum, Tanja und Becker, Jürgen: „Automated Replacement of State-Holding Flip-Flops to Enable Non-Volatile Checkpointing“. In: *2023 IEEE Nordic Circuits and Systems Conference (NorCAS)*. 2023, S. 1–7. doi: 10.1109/NorCAS58970.2023.10305469.

[Kre24a] Kreß, Fabian; El Annabi, El Mahdi; Hotfilter, Tim; Hoefer, Julian; Harbaum, Tanja und Becker, Juergen: „Automated Deep Neural Network Inference Partitioning for Distributed Embedded Systems“. In: *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 2024, S. 39–44. doi: 10.1109/ISVLSI61997.2024.00019.

[Kre24b] Kreß, Fabian; Sidorenko, Vladimir; Topko, Iuliia; Unger, Kai; Harbaum, Tanja und Becker, Jürgen: „VHDL Crash Course: A Multimedia-Based Teaching Approach“. In: *2024 IEEE 3rd German Education Conference (GECon)*. 2024, S. 1–6. doi: 10.1109/GECon62014.2024.10734007.

[Kre24c] Kreß, Fabian; Serdyuk, Alexey; Kobsar, Denis; Hotfilter, Tim; Hoefer, Julian; Harbaum, Tanja und Becker, Jürgen: „LOTTA: An FPGA-based Low-Power Temporal Convolutional Network Hardware Accelerator“. In: *2024 IEEE 37th International System-on-Chip Conference (SOCC)*. 2024, S. 126–131. doi: 10.1109/SOCC62300.2024.10737863.

[Kre25] Kreß, Fabian; Hoefer, Julian; Lin, Qiushi; Schmidt, Patrick; Harbaum, Tanja und Becker, Jürgen: „Deep Neural Network Inference Partitioning in Embedded Analog-Digital Hybrid Systems“. In: *2025 26th International Symposium on Quality Electronic Design (ISQED)*. in press. 2025.

[Mas18] Masing, Leonard; Srivatsa, Akshay; Kreß, Fabian; Anantharajaiah, Nidhi; Herkersdorf, Andreas und Becker, Jürgen: „In-NoC Circuits for Low-Latency Cache Coherence in Distributed Shared-Memory Architectures“. In: *2018 IEEE 12th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. 2018, S. 138–145. doi: 10.1109/MCSoC2018.2018.00033.

[Hot21] Hotfilter, Tim; Hoefer, Julian; Kreß, Fabian; Kempf, Fabian und Becker, Juergen: „FLECSim-SoC: A Flexible End-to-End Co-Design Simulation Framework for System on Chips“. In: *2021 IEEE 34th International System-on-Chip Conference (SOCC)*. 2021, S. 83–88. doi: 10.1109/SOCC52499.2021.9739212.

[Les21] Lesniak, Fabian; Kreß, Fabian und Becker, Jürgen: „Transparent Near-Memory Computing with a Reconfigurable Processor“. In: *Applied Reconfigurable Computing. Architectures, Tools, and Applications*. Hrsg. von Derrien, Steven; Hannig, Frank; Diniz, Pedro C. und Chillet, Daniel. Cham: Springer International Publishing, 2021, S. 221–231.

[Hot22a] Hotfilter, Tim; Kreß, Fabian; Kempf, Fabian; Becker, Jürgen; De Haro, Juan Miguel; Jiménez-González, Daniel; Moretó, Miquel; Álvarez, Carlos; Labarta, Jesús und Baili, Imen: „Towards Reconfigurable Accelerators in HPC: Designing a Multipurpose eFPGA Tile for Heterogeneous SoCs“. In: *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2022, S. 628–631. doi: 10.23919/DATE54114.2022.9774716.

[Hot22b] Hotfilter, Tim; Kreß, Fabian; Kempf, Fabian; Becker, Jürgen und Baili, Imen: „Data Movement Reduction for DNN Accelerators: Enabling Dynamic Quantization Through an eFPGA“. In: *2022 IEEE Computer Society*

Annual Symposium on VLSI (ISVLSI). 2022, S. 371–372. DOI: 10.1109/ISVLSI54635.2022.00082.

[Kem22] KEMPF, Fabian; HOEFER, Julian; KREß, Fabian; HOTFILTER, Tim; HARBAUM, Tanja und BECKER, Juergen: „Runtime Adaptive Cache Checkpointing for RISC Multi-Core Processors“. In: *2022 IEEE 35th International System-on-Chip Conference (SOCC)*. 2022, S. 1–6. DOI: 10.1109/SOCC56010.2022.9908110.

[Hoe23a] HOEFER, Julian; KEMPF, Fabian; HOTFILTER, Tim; KREß, Fabian; HARBAUM, Tanja und BECKER, Jürgen: „SiFI-AI: A Fast and Flexible RTL Fault Simulation Framework Tailored for AI Models and Accelerators“. In: *Proceedings of the Great Lakes Symposium on VLSI 2023*. GLSVLSI '23. Knoxville, TN, USA: Association for Computing Machinery, 2023, S. 287–292. DOI: 10.1145/3583781.3590226. URL: https://doi.org/10.1145/3583781.3590226.

[Hoe23b] HOEFER, Julian; HOTFILTER, Tim; KREß, Fabian; QIU, Chen; HARBAUM, Tanja und BECKER, Juergen: „A Hardware-Aware Sampling Parameter Search for Efficient Probabilistic Object Detection“. In: *Computer Vision Systems*. Hrsg. von CHRISTENSEN, Henrik I.; CORKE, Peter; DETRY, Renaud; WEIBEL, Jean-Baptiste und VINCZE, Markus. Cham: Springer Nature Switzerland, 2023, S. 299–309.

[Hot23a] HOTFILTER, Tim; SCHMIDT, Patrick; HÖFER, Julian; KREß, Fabian; HARBAUM, Tanja und BECKER, Juergen: „An Analytical Model of Configurable Systolic Arrays to Find the Best-Fitting Accelerator for a given DNN Workload“. In: *Proceedings of the DroneSE and RAPIDO: System Engineering for Constrained Embedded Systems*. RAPIDO '23. Toulouse, France: Association for Computing Machinery, 2023, S. 73–78. DOI: 10.1145/3579170.3579258. URL: https://doi.org/10.1145/3579170.3579258.

[Hot23b] HOTFILTER, Tim; HOEFER, Julian; KREß, Fabian; KEMPF, Fabian; KRAFT, Leonhard; HARBAUM, Tanja und BECKER, Jürgen: „A Hardware-Centric Approach to Increase and Prune Regular Activation Sparsity in CNNs“. In: *2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. 2023, S. 1–5. DOI: 10.1109/AICAS57966.2023.10168566.

[Hot23c] HOTFILTER, Tim; HOEFER, Julian; MERZ, Philipp; KREß, Fabian; KEMPF, Fabian; HARBAUM, Tanja und BECKER, Jürgen: „Leveraging Mixed-Precision CNN Inference for Increased Robustness and Energy Efficiency“. In: *2023 IEEE 36th International System-on-Chip Conference (SOCC)*. 2023, S. 1–6. DOI: 10.1109/SOCC58585.2023.10256738.

[Ser23] SERDYUK, Alexey; KREß, Fabian; HIEGLE, Micha; HARBAUM, Tanja; BECKER, Jürgen; IMBERT, Florent; SOULLARD, Yann; TAVENARD, Romain; ENQUETIL, Eric; BARTH, Jens und KÄMPF, Peter: „Towards the on-device Handwriting Trajectory Reconstruction of the Sensor Enhanced Pen“. In: *2023 IEEE 9th World Forum on Internet of Things (WF-IoT)*. 2023, S. 1–6.

[Sta23] STAMMLER, Matthias; SIDORENKO, Vladimir; KREß, Fabian; SCHMIDT, Patrick und BECKER, Jürgen: „Context-Aware Layer Scheduling for Seamless

Neural Network Inference in Cloud-Edge Systems“. In: *2023 IEEE 16th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. 2023, S. 97–104. DOI: 10.1109/MCSoC60832.2023.00022.

[Bal24] Balaskas, Konstantinos; Khdr, Heba; Bakr Sikal, Mohammed; Kreß, Fabian; Siozios, Kostas; Becker, Jürgen und Henkel, Jörg: „Heterogeneous Accelerator Design for Multi-DNN Workloads via Heuristic Optimization“. In: *IEEE Embedded Systems Letters* 16.4 (2024), S. 317–320. DOI: 10.1109/LES.2024.3443628.

[Har24a] Harbaum, Tanja u. a.: „KIHT: Kaligo-Based Intelligent Handwriting Teacher“. In: *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2024, S. 1–6. DOI: 10.23919/DATE58400.2024.10546623.

[Har24b] Harbaum, Tanja; Topko, Iuliia; Serdyuk, Alexey; Fürst-Walter, Iris; Kreß, Fabian und Becker, Jürgen: „HW/SW Co-Design for Integrated AI Systems: Challenges, Use Cases and Steps Ahead“. In: *3rd Workshop on Deep Learning for IoT (DL4IoT-2024)*. 2024.

[Hoe24] Hoefer, Julian; Gauß, Michael; Adams, Manuela; Kreß, Fabian; Kempf, Fabian; Karle, Christian; Harbaum, Tanja; Barth, Andreas und Becker, Juergen: „A Challenge-Based Blended Learning Approach for an Introductory Digital Circuits and Systems Course“. In: *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2024, S. 1–5. DOI: 10.1109/ISCAS58744.2024.10557955.

[Ser24] Serdyuk, Alexey; Kreß, Fabian; Topko, Iuliia; Harbaum, Tanja; Becker, Jürgen; Hamann, Tim und Kämpf, Peter: „Improving Online Handwriting Trajectory Reconstruction Based on Temporal Convolutional Networks“. In: *2024 Sensor Data Fusion: Trends, Solutions, Applications (SDF)*. 2024, S. 1–8. DOI: 10.1109/SDF63218.2024.10773805.

[Top25] Topko, Iuliia; Kreß, Fabian; Harbaum, Tanja und Becker, Juergen: „General Compilation and Mixed-Precision Partitioning: A Combined Approach for Adaptive On-Device Learning“. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2025). under review.

[Sch25] Schmidt, Patrick; Kreß, Fabian; Harbaum, Tanja und Becker, Jürgen: „DSEParted: Co-Optimization of NPU Architectures and Neural Network Partitioning“. In: *Proceedings of the Great Lakes Symposium on VLSI 2025*. GLSVLSI '25. under review. New Orleans, LA, USA: Association for Computing Machinery, 2025.

Studentische Arbeiten

[Buy21] Buyer, Simon: „Hypervisor-basiertes Framework zur Evaluation neuartiger Speichertechnologien“. Bachelor's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2021.

[Den21] Deng, Zihao: „An Emulation Framework to Evaluate NVM-based Flip-flops in Processor Architectures“. Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2021.

[He22a] He, Xinnan: „Design and Evaluation of RFET In-Memory-Computing Architectures and Programming Toolflow“. Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2022.

[He22b] He, Zhuofan: „Entwicklung einer FPGA Synthese-Toolchain zur automatisierten Integrierung hybrider Flip-Flops“. Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2022.

[Lim22] Limame, Amani: „Automatisierte Optimierung für CNN-Autoencoder im Bereich IoT-Objekterkennung“. Bachelor's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2022.

[Yua22] Yuan, Yueming: „A Low-Power RISC-V Core for Tiny Machine Learning“. Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2022.

[Zha22a] Zhamoliddinov, Zhasur: „Modellierung eines MTJ-basierten Flip-Flops für registerbasiertes Checkpointing“. Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2022.

[El-23] El-Annabi, El Mahdi: „Development of an Automated Framework for determining beneficial CNN Inference Partitioning“. Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2023.

[Bas23] Bashkuev, Vladimir: „Latency-optimised optical link for multi-FPGA neural network implementation“. Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2023.

[Bus23] Buschmann, Ilka: „KI-basierte Online Trajektorienrekonstruktion in einem digitalen Stift“. Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2023.

[Sun23b] Sundaram, Akilandeeswari Shanmuga: „Modeling embedded, energy-efficient systems in future IoT applications“. Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2023.

[Wal23] Waldmann, Disnebio: „Hardware/Software Co-Design einer Ultra-Low Power RISC-V Plattform für online Handschrifterkennung“. Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2023.

[Yil23] Yilmaz, Bahadirhan: „Systematische Performanzanalyse von eingebetteten Systemen für IoT Anwendungen“. Bachelor's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2023.

[Kob24] Kobsar, Denis: „Entwicklung eines eingebetteten Hardware-Beschleunigers zur effizienten Ausführung von TCNs“. Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2024.

Abbildungen

Tabellen

Abkürzungsverzeichnis

ADAS	Advanced Driver-Assistance System
ADC	Analog-to-Digital Converter
ALU	Arithmetisch-Logische Einheit
ASIC	Application Specific Integrated Circuit
BLE	Bluetooth Low Energy
BLIF	Berkeley Logic Interchange Format
BRAM	Block RAM
CER	Character Error Rate
CFS	Custom Function Subsystem
CMOS	Complementary Metal Oxide Semiconductor
CNN	Convolutional Neural Network
CONV	Convolutional Layer
CPU	Central Processing Unit
CTC	Connectionist Temporal Classification
DAG	gerichteter azyklischer Graph
DDR	Double Data Rate
DLA	Deep Learning Accelerator
DMEM	Data Memory
DNN	Deep Neural Network
DSP	Digitaler Signal-Prozessor
ECU	Electronic Control Unit
EDA	Electronic Design Automation
EDP	Energy-Delay-Produkt
FF	Flip-Flop
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GA	Genetischer Algorithmus

GPU	Graphics Processing Unit
HLS	High-Level Synthese
HPC	High-Performance Computing
IC	Integrierter Schaltkreis
IMU	Inertial Measurement Unit
IoT	Internet-of-Things
IP	Intellectual Property
ISA	Instruction Set Architecture
KI	Künstliche Intelligenz
LC	Logic Cell
LM	Language Model
LSTM	Long Short-Term Memory
LUT	Look-Up Table
MAC	Multiply-Accumulate
MFF	Magnetischer Flip-Flop
ML	Machine Learning
MPSoC	Multiprozessor-System-on-Chip
MRAM	Magnetoresistive Random Access Memory
MSE	Mean Squared Error
MTK	Magnetische Tunnel-Kontakt
MVM	Matrix-Vector-Multiplikation
NAS	Neural Architecture Search
NoC	Network-on-Chip
NVSRAM	Non-Volatile Static Random-Access Memory
ONNX	Open Neural Network Exchange
PCM	Phase Change Random Access Memory
PHY	Physical layer
PIM	Processing-In-Memory
PLL	Phase-Locked Loop
PTQ	Post Training Quantization
QAT	Quantization-aware Training

RAM	Random-Access Memory
RNN	Recurrent Neural Network
RRAM	Resistive Random Access Memory
RTL	Register Transfer Level
SAF	Stuck-At Fault
SoC	System-on-Chip
SPI	Serial Peripheral Interface
SRAM	Static Random-Access Memory
TCN	Temporal Convolutional Network
UCIe	Universal Chiplet Interconnect Express