

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY,
AND MEDICINE

DEPARTMENT OF ELECTRICAL AND ELECTRONIC
ENGINEERING

**ATHEENA:
Automated Toolflow for Hardware
Early-Exit Network Acceleration**

PhD Candidate:

Benjamin Mark Biggs

Supervisors:

Prof. G. CONSTANTINIDES

Prof. C-S. BOUGANIS

Internal Examiner:

Prof. P. CHEUNG

External Examiner:

Prof. R. SHAFIK

Submitted in part fulfilment of the requirements for the degree of Doctor of Philosophy in
Electrical and Electronic Engineering of Imperial College London and the Diploma of
Imperial College London.

Copyright Declaration

The copyright of this thesis rests with the author, Benjamin Mark Biggs. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution-Non Commercial 4.0 International Licence (CC BY-NC).

Under this licence, you may copy and redistribute the material in any medium or format. You may also create and distribute modified versions of the work. This is on the condition that: you credit the author and do not use it, or any derivative works, for a commercial purpose.

When reusing or sharing this work, ensure you make the licence terms clear to others by naming the licence and linking to the licence text. Where a work has been adapted, you should indicate that the work has been changed and describe those changes.

Please seek permission from the copyright holder for uses of this work that are not included in this licence or permitted under UK Copyright Law.

Copyright © 2025 Benjamin Biggs.

Statement of Originality

I, Benjamin Mark Biggs, declare that the work presented in this thesis is work I have produced and that any other work has been suitably referenced.

Abstract

The continued need for improvements in accuracy, throughput, and efficiency of Deep Neural Networks has resulted in a multitude of static parameter reduction methods, like quantisation and pruning, which leverage the custom architectures possible on FPGAs. However, the potential of these solutions is already well exploited, reaching the limits of what can be achieved by reducing the number or size of the parameters while still maintaining accuracy. We propose a shift of focus to *input-dependent* computation to improve efficiency and reduce the average compute required for inference. Early-Exit (EE) networks have become an increasingly popular way to implement dynamic parameter reduction by varying network depth, essentially customising the computation level according to the difficulty of an input at run-time.

We create *Automated Toolflow for Hardware Early-Exit Network Acceleration* (ATHEENA), an automated, open-source CNN-to-FPGA toolflow which utilises the probability of samples exiting early from EE networks to optimally allocate the limited resources of an FPGA to different sections of the network. This ultimately results in improved throughput. The toolflow uses the data-flow model of the existing fpgaConvNet tool, extended to support Early-Exit networks, as well as Design Space Exploration (DSE) to optimise the generated streaming architecture hardware with the goal of increasing throughput/reducing area while maintaining accuracy. To this end, we incorporate abstracted hardware models, based on Queueing Theory, to aid the DSE with a more accurate analysis of performance and resource requirements. This improves the robustness of the accelerator.

Experimental results on three different networks demonstrate a throughput increase of $2.00\times$ to $3.12\times$ compared to an optimised baseline network implementation with no early exits. Additionally, the toolflow can achieve a throughput matching the same baseline with as low as 48% of the resources the baseline requires.

Acknowledgements

I have many many people that deserve acknowledgement, a tremendous thank you to you all. Firstly, I would like to thank my supervisors George Constantinides and Christos-Savvas Bouganis for going above and beyond in supporting me both academically and in a personal capacity. I have been very fortunate to have their guidance on all matters theoretical and technical in relation to my research, and I will always appreciate their continued faith in me, even when I had none in myself. Thank you very much for taking a chance with me as a PhD student and being my academic dads!

A thank you to my examiners: Professor Peter Cheung and Professor Rishad Shafik for reading my thesis and engaging with me in a thoroughly interesting and in-depth examination.

Next I would like to thank my collaborators. Luigi Rinaldi is an incredibly bright student and he was a joy to supervise/collaborate with during his UROP. Anastasios Dimitriou is my fellow EE & FPGAs researcher and is probably one of the only other people that will understand my thesis. I'm very glad we got to do our little workshop paper together.

Thank you to the cornerstone of the Circuits And Systems (CAS) group, Wiesia Hsissen for being an administrative wizard and always having a kind word of reassurance to give.

Speaking of the CAS group, I must give thanks to the members of the old guard (Doctors all): Erwei, Deetz, Adi, Jianyi, Alex Dalton, Ben Chua, and Yann. Thanks for your coffee brewing expertise, moral support, kind words, and technical expertise. And of course Ian McInerney for supervising me through my undergrad FYP and ultimately getting me to the point of doing a PhD.

A big thank you to the fpgaConvNet gang: Alex MC, Zhewen, and Petros. Thank you so much for your technical knowledge, your patience when stuff wasn't working, and for being great fun at conferences. Much of the research in this thesis would not have been possible without your help!

Thank you to Sam and Marta for being shining examples of how to do academia right, for being a joy to hang out with, and for your support. Thank you to Michalis for being a good friend and the God of servers, so many PhDs would be impossible without you!

A huge thank you to my fellow CAS members for providing a great research environment: Cheng, Mingzhu, Quentin, Victor, Zehui, Guoxuan, Ebby, Keran, and Tony. And another Thank you to the

new gang Violet, Jeffrey and Cano. I hope you enjoy the rest of your time in CAS!

A massive thank you to my friends from undergrad (a.k.a. The Bridge Crew) and particularly memelord Tomasz for being a great friend, always letting me rant and REEE, and geeking out with me on kewn tech things.

Thanks to the IC Dance team for being easily one of my favourite things about Imperial College. Matt, Lucy, and Devon, I'm so glad we were on the team together, you are all such fantastic people, and I'm truly thankful we are such good mates.

Thanks to my housemates (and good friends): Chris, Alice, and Jack for being great to live with for most of my PhD and for all the fun board games and great food we've shared!

An enormous thank you to Pana and Colin for being so incredibly generous with their time (and delicious food!) and especially for putting a roof over my head.

I could not have done this without the wonderful and limitless support of my family, I love you all. Nanny, you've always championed education and are the most compassionate person I know. Granny, you've always encouraged me and been an amazing example of getting things done. Grandpa and Dad, you are both so intelligent and I am so grateful you instilled in me a fascination with science at a young age. Keep the articles coming! Bex, El Sistro, thank you for being the best sibling I could ask for! You've always been in my corner, you are super smart, and long may all our in jokes continue. Mama, I can't thank you enough for truly being with me every step of the way. You have encouraged me and supported me more than anyone, and I am so lucky to have you as my mum.

My dearest Zoë, I really can't put into words how fortunate I am to have had your unending love and support through, not only my PhD but most of my undergrad too! Thank you for always being there with a hug and a cup of tea whenever the PhuDding brain got too much. Like my Supervisors, my Mum, and my Sis, you have always believed in me more than myself and I am so thankful. You are a genius writer and academic wizard and I'm so lucky that you are my dance partner and my love.

P.S. Thanks to the Monterey Bay Otters, Bol of the Stump, and the wonderful team at iCope NHS camden for repairing my mental health.

Contents

Copyright Declaration	2
Statement of Originality	3
Abstract	4
Acknowledgements	5
List of Figures	12
List of Tables	13
1 Introduction	14
1.1 Research Aim & Objectives	17
1.2 Research Contributions	18
1.2.1 Code Repositories	21
1.2.2 Publications	21
1.3 Dissertation Outline	22
2 Background & Literature Review	23
2.1 Input-dependent Computation in DNNs	23
2.1.1 Early-Exit Networks	27
2.2 Modelling Probabilistic Systems: Queueing Theory	33

2.3	FPGAs	35
2.4	DNN compilers for FPGAs	36
2.4.1	fpgaConvNet	38
2.5	Dynamic DNNs on FPGAs	40
2.5.1	Toolflows	43
2.6	Summary	45
3	Method for the Efficient Acceleration of Early-Exit CNNs on FPGAs	47
3.1	Introduction	47
3.2	Scaling Resource Allocation by Exit Probability	48
3.2.1	Multiple Early-Exit Stages	55
3.3	Summary	56
4	ATHEENA: A Toolflow for Hardware Early-Exit Network Automation	58
4.1	Introduction	58
4.2	Front-end Representation	59
4.2.1	ONNX Conversion	61
4.2.2	Early-Exit Profiler	62
4.3	Early-Exit Network Layers: Hardware Templates	62
4.3.1	Exit (Softmax) Decision Layer	63
4.3.2	Conditional Buffer Layer	64
4.3.3	Split Layer	66
4.3.4	Exit Merge Layer	66
4.4	HLS Limitations & Parallel Compilation	67
4.5	Experimental Results	68
4.5.1	BranchyNet: Hardware Experimental Study	69

4.5.2	Benchmarking Results	76
4.6	Summary	77
5	Queueing Theory for Throughput Estimation with Finite Buffers	78
5.1	Introduction	78
5.2	Accelerator Abstraction	80
5.2.1	Analytical Models	82
5.3	Verification: Accuracy & Computational Analysis	85
5.4	Resource Allocation Under Finite Buffer Constraints	89
5.5	Finite Buffer Approximation Behaviour	92
5.6	Summary	95
6	Improved Resource and Throughput Estimation for Early-Exit CNN Accelerators on FPGAs	96
6.1	Introduction	96
6.2	Toolflow Improvements	97
6.2.1	Updated Resource Modelling	97
6.3	Experimental Results	99
6.3.1	Effect of Queueing Theory	100
6.3.2	Returning to BranchyNet Case-Study	103
6.3.3	Benchmarking Results	107
6.4	Addressing Buffer Placement	108
6.4.1	Benefits of Alternative Placement	109
6.4.2	Disadvantages of Altering Placement	109
6.4.3	Buffer Placement Experiment	110
6.5	Summary	111

7	Improving the Early-Exit Decision Component and Corresponding Hardware	112
7.1	Introduction	112
7.2	Improving the Decision Component	115
7.3	New Decision Component Evaluation	119
7.3.1	Examining Raw Maximum Value Distributions	119
7.3.2	Examining Confidence Metric Value Distributions	122
7.3.3	Applying Classifier Measures: AUROC & AUPRC	123
7.3.4	Decision Component Hardware Statistics	132
7.4	Summary	134
8	Conclusion	136
8.1	Observations	137
8.2	Limitations & Future Directions	139
8.3	Summary	141
	References	158

List of Figures

1.1	High-level overview of the ATHEENA toolflow.	19
1.2	Dissertation outline	22
2.1	Simplified taxonomy of DDNNs.	24
2.2	Generic Early-Exit Network Architecture	27
2.3	Confidence metric distribution examples.	29
2.4	Simple queueing system.	32
2.5	Simple Markov chain representation of a queueing system.	33
2.6	Working principle for the different approaches to Decision Sub-Networks [82]. . .	42
3.1	High-level diagram of control-flow hardware attached to the CNN layers.	50
3.2	A visual representation of combining TAP functions.	52
3.3	Scaling the throughput of the second stage.	53
3.4	Deviations in design and run-time throughput.	55
4.1	A visual representation of the ATHEENA toolflow.	60
4.2	New control-flow Early-Exit Hardware layer templates.	63
4.3	New data-flow Early-Exit Hardware layer templates.	66
4.4	Slightly modified version of B-LeNet [22] for fpgaConvNet.	70
4.5	Comparison of baseline and ATHEENA performance.	71

5.1	Queueing system abstraction of accelerator hardware.	81
5.2	Finite queue Markov chain representation.	83
5.3	Simulated and predicted curves for throughput fraction.	88
5.4	Determining the mean second stage throughput.	90
5.5	Queueing Theory-informed resource allocation and TAP combination.	91
5.6	Blocking probability p_B against traffic intensity ρ and queue size k , for a fixed p	94
5.7	Estimation of the adjusted TAP function with deviations in run-time blocking probability.	94
6.1	A visual representation of the improved ATHEENA toolflow.	98
6.2	Demonstration of Queueing Theory benefits.	102
6.3	Comparison of Baseline and updated ATHEENA performance.	105
7.1	Impact of the Early-Exit branch.	113
7.2	Current SoftMax Decision Component.	116
7.3	Raw layer value distribution.	121
7.4	Processed confidence metric value distributions.	124
7.5	ROC curves for each confidence metric.	127
7.6	The Precision-Recall (PR) Curve for each confidence metric.	129
7.7	The exit rate at a given threshold value for each confidence metric.	131

List of Tables

2.1	Summary of Toolflows.	43
4.1	Resource comparison for implemented Baseline Vs ATHEENA.	74
4.2	Resource overhead of the Early-Exit compared to network backbone.	74
4.3	Comparison against software BranchyNet.	75
4.4	Throughput improvements for designs generated by ATHEENA.	76
5.1	Quantitative view of the different Queueing Theory-based models.	86
6.1	Resource comparison for implemented Baseline Vs ATHEENA.	107
6.2	Throughput improvement on benchmark networks.	108
7.1	Evaluating decision component hardware resources.	132
7.2	Evaluating decision component hardware performance.	134

CHAPTER

1

Introduction

Deep Neural Networks (DNNs) have become increasingly popular due to their performance across a broad range of tasks, particularly in computer vision [1] and natural language processing [2]. There are many factors responsible for this performance, including: high parameter, high capacity networks; increasingly large datasets; and powerful computational engines capable of executing these workloads. The search for ever-increasing accuracy has typically lead to the creation of larger networks which require increased computing power. This computing power can be obtained through the use of next-generation CPUs and GPUs but this results in costly systems, both in financial and energy terms [3], [4]. A potential solution is the creation of custom, application-specific hardware, tailored to a given DNN’s workload. We can see examples of this in practice with Google [5], and Graphcore [6] with the caveat that developing custom ICs has a design, verification, and manufacturing cost; moreover, the DNN designer becomes locked into the hardware architecture, potentially missing out on the fast-paced developments in the DNN space. With this in mind, we turn to Field-Programmable Gate Arrays (FPGAs).

FPGAs are low power, flexible devices key for tailoring accelerators to differing DNN workloads by leveraging custom precision and highly parallel computation. Low power consumption, when compared to GPUs, is necessary when targeting embedded systems or IoT devices. The flexible routing, and composition of programmable resources within the device, enable the im-

plementation of different DNN types, e.g., Convolutional Neural Networks (CNN). CNNs have many applications, and are often the preferred choice to Vision Transformers due to their relative computational efficiency in computer vision and image classification tasks [7]. In spite of this, many of the top performing networks suffer from large workloads that prohibit their execution on FPGAs and edge devices with limited resources. In order to accelerate CNNs using FPGAs, inference workload and memory footprint need to be reduced whilst maintaining task performance. To this end, there have been a wide variety of static parameter reduction methods derived to facilitate the execution of these networks and mitigate the resource limitations of many target devices [8]. These methods ¹ include pruning [9]–[11], quantisation [12]–[16], and knowledge distillation [17], [18]. We refer to these methods as static because they are applied, in one or more passes, *prior* to deployment. The successful application of static parameter reduction implies that a full-scale version of the network has some level of parameter redundancy, exploitable across the majority of the data set, resulting in minimal accuracy degradation.

Parallel to network compression with post-training parameter reduction, researchers are altering network architectures through hand-crafted design [19] and Neural Architecture Search (NAS) [20], [21]. A key aim of the resultant CNN architectures is to reduce network footprint for low resource settings like embedded systems and mobile devices. Taken to its natural conclusion, these new architectural approaches will cause newer CNN models to be less redundant, decreasing the efficacy of static parameter reduction methods.

This is where *input-dependent* computing can take over. The fundamental concept is that a given input sample has an associated difficulty that correlates to the computation required to achieve an accurate result. If we can alter the level of computation on a per input basis, we can achieve *dynamic* parameter reduction. We focus on depth-based dynamic methods, specifically the Early-Exit approach, due to their performance when applied to classification

¹There are dynamic parameter reduction implementations of both pruning and quantisation as well, see Section 2.5.1.

CNNs [22] and the intuitive reasoning behind their effectiveness. Take an example input A . This input is ‘easy’ so can be accurately classified based on simple features derived earlier in the CNN. However, input B is more challenging so requires the more refined features and greater receptive field of deeper layers, resulting in a higher computation requirement. By dynamically adapting the computation based on the perceived difficulty, it is possible to reduce the average compute required by inference for the low overhead of a confidence calculation. This confidence calculation provides an accuracy estimate for a task result at different depths. The user provides a threshold with which to compare the accuracy estimates in order to determine the results that can confidently exit early. Since the ‘difficult’ samples can continue through the full network, there is often little to no decrease in accuracy. In some cases, accuracy can improve over the baseline network [23], a result of ‘over-thinking’². Early-Exit approaches have seen application across a wide range of tasks [24] in the computer vision space as they expose a run-time tunable trade-off between accuracy and computation.

We have outlined the core concepts of this dissertation: FPGA-based acceleration for DNN workloads, and depth-based dynamic DNNs as a method for reducing average computation. This dissertation explores how we can best harness the dynamic nature of these DNNs in order to create efficient accelerators on FPGA devices. The flexibility of an FPGA results in an expansive design space that cannot be explored exhaustively in a reasonable time-frame. The introduction of input-dependency adds another dimension of possible designs. We take a programmatic, step by step approach to apportioning the limited resources of a given FPGA in order to balance the stages of the accelerator with respect to the profiled behaviour of a given network. This simplifies the process for a DNN expert that wants to accelerate throughput-oriented applications that are subject to latency constraints prohibiting device reconfiguration at run-time. This results in a toolflow that can take a pre-trained Early-Exit CNN model and

²A phenomenon observed in which a correct early result is confidently altered to an incorrect result deeper in the network.

produce a FPGA-based accelerator that outperforms an equivalent standard CNN. The main thesis of this dissertation is as follows:

Given a pre-trained Early-Exit CNN, we can utilise the properties of depth-based Dynamic Deep Neural Networks by optimally allocating FPGA resources, creating an automated toolflow that yields high throughput accelerators for Early-Exit CNNs.

1.1 Research Aim & Objectives

The primary aim of this dissertation is to develop a toolflow that can improve the inference throughput of a pre-trained, Early-Exit CNN by automatically designing and implementing custom FPGA-based accelerators. To facilitate this aim we outline the following objectives:

1. Develop an approach for mapping an Early-Exit CNN to FPGA. This should include the creation of the FPGA hardware components that enable the depth-based dynamic execution features of the Early-Exit CNN.
2. Determine a method that maximises accelerator throughput by leveraging the probabilistic use of the different paths an input sample can take through an Early-Exit CNN.
3. Increase the accuracy of the methods by which resource consumption and throughput are estimated in order to improve the exploration of the FPGA-based accelerator design space.
4. Address the resource constraints of the target device, reducing the reliance on unrealistic assumptions.
5. Build more rigorous probabilistic modelling in to the design stage of the toolflow.

6. Analyse the method used for determining confidence in a result that exits early. Use this to determine the impact of reduced precision and approximation in the effectiveness of the corresponding Decision component.
7. Build template FPGA hardware that make use of approximation in order to reduce the resource consumption and latency of the Decision component. Subsequently improving the throughput of the accelerator and reducing the resource overhead of the early-exit branches.

1.2 Research Contributions

The research contributions of this dissertation are summarised below:

- A procedure for obtaining the probability profiles of a pre-trained Early-Exit CNN. These probability profiles are estimations of the probability a given input sample has of exiting the network early. This provides us with network statistics that enable us to carry out objective 2.
- A methodology for utilising probability profiles to allocate FPGA resources to the different stages of an Early-Exit (EE) CNN in order to explore the trade-off between throughput and area³. This approach contributes to objectives 1 & 2, allowing us to approach the problem of efficiently mapping Early-Exit networks to FPGAs.
- A set of FPGA-friendly components for building Early-Exit networks on FPGAs, building on the open-source fpgaConvNet [25] toolflow. This contributes to our primary aim as well as objective 1, addressing the problem of a lack of support for conditional inference in open-source tools.

³We note that ‘resources’ and ‘area’ are used interchangeably throughout this dissertation because the FPGA-based accelerator designs have identical frequencies.

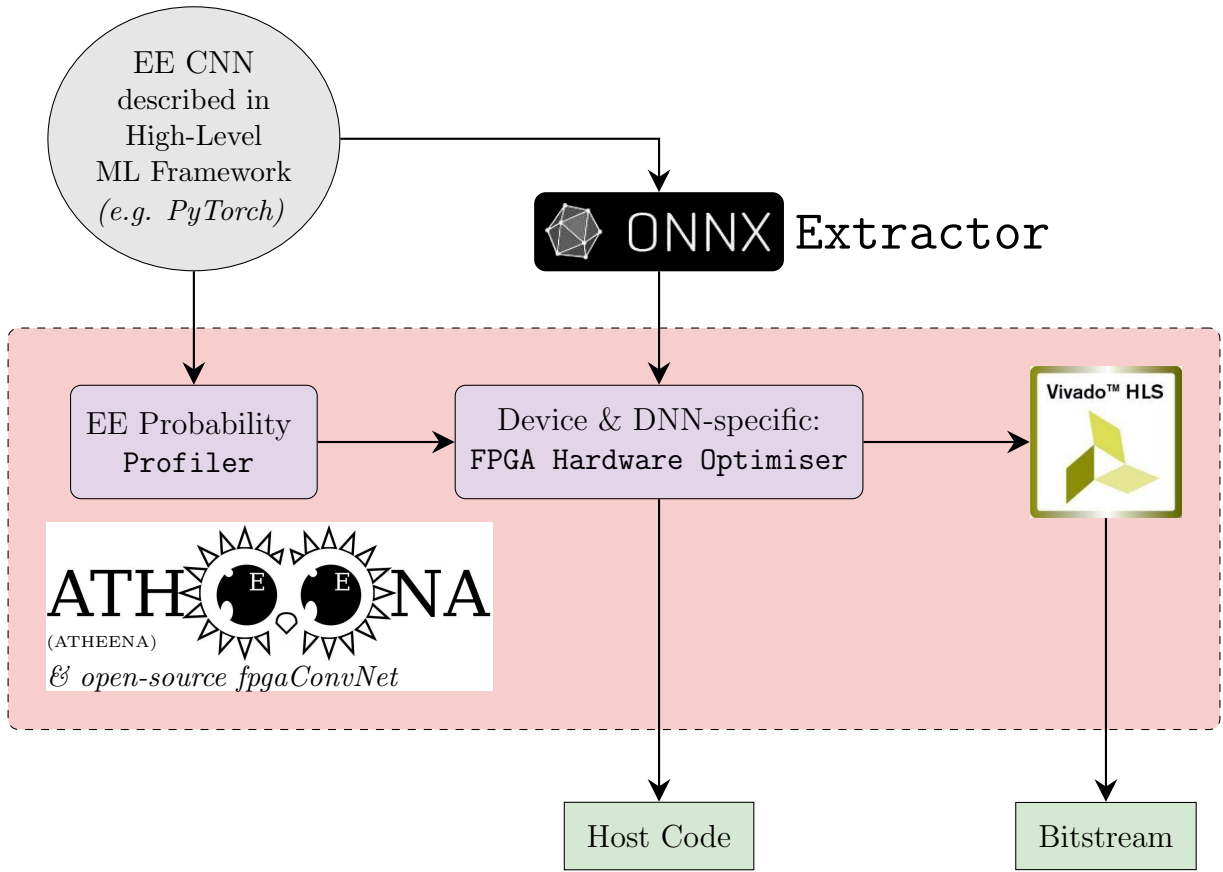


Figure 1.1: High-level overview of the ATHEENA toolflow.

- The ATHEENA automated toolflow (Figure 1.1) – which utilises profiling probabilities and translates Early-Exit CNNs from an ONNX-based representation into custom C++ code suitable for compilation using Vivado HLS. This toolflow addresses the primary aim of the thesis and provides a mechanism for users of Deep Learning to accelerate workloads in resource-constrained environments.
- Development of techniques to model the throughput of the ATHEENA-generated accelerators based on probabilistic models derived from Queueing Theory. These models are verified by comparison with event-driven simulations and allow the toolflow to estimate the effect of finite on-chip buffers between network exit stages. This contribution tackles objectives 3, 4, & 5, establishing Queueing Theory as an appropriate method for modelling dynamic DNN execution.
- An analysis of the accuracy and speed of the aforementioned techniques, as part of the

Design Space Exploration (DSE) procedure for generating Early-Exit CNN accelerators. While not directly contributing to an objective, this analysis is important in creating a toolflow with accurate estimation and reasonable compilation time.

- Updated resource and throughput modelling that incorporates the use of finite buffers. This improves the estimation of the trade-off between resource usage and throughput, and addresses objectives 3 & 5.
- A detailed analysis of the effects of changes in the precision and calculation of the confidence metric used for determining whether or not an early-exit occurs. This results in the design of an improved decision component, addressing objectives 6 & 7. This contributes an efficient open-source component for calculating the confidence Decision as well as a thorough understanding of the behaviour of the confidence calculation.

1.2.1 Code Repositories

The code supporting this research is contained within two open-source repositories:

- [ATHEENA](#) – documentation and toolflow for Early-Exit CNN acceleration. This was provided as an artefact in support of the ATHEENA conference paper, and the results were verified by an external academic.
- [Early-Exit Net](#) – training, testing, and experimentation with Early-Exit networks.

1.2.2 Publications

- *ATHEENA: A Toolflow for Hardware Early-Exit Network Automation* [26] – published at the 31st International Symposium on Field-Programmable Custom Computing Machines (FCCM) 2023.
- *FPGA Acceleration of Dynamic Neural Networks: Challenges and Advancements* [27] – published at IEEE International Conference on Omni-layer Intelligent Systems (COINS) 2024.
- Journal Extension of the ATHEENA FCCM work. Currently under submission to IEEE Journal: Transactions on Computer Aided Design.

1.3 Dissertation Outline

Technical Chapters 3–6 follow the main developments of the toolflow. Chapter 7 is a parallel, improvement of the Decision component hardware, backed by experimental analysis.

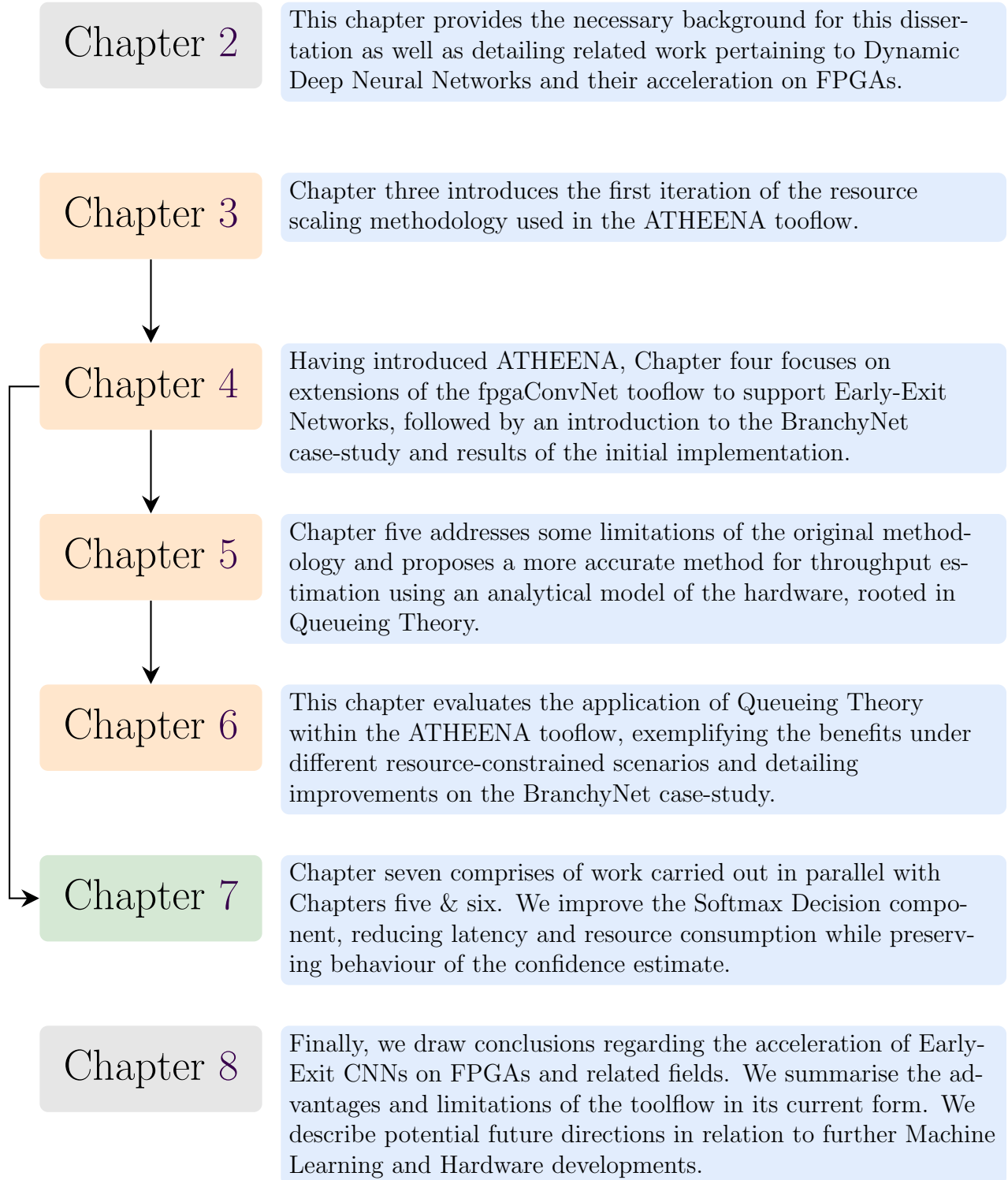


Figure 1.2: *Dissertation outline.*

CHAPTER

2

Background & Literature Review

The following chapter explores input-dependent, dynamic DNN architectures with a focus on Early-Exit networks. We provide a brief explanation of Queueing Theory and its potential for modelling the probabilistic behaviour of dynamic systems. We introduce FPGAs as the target device of choice and state-of-the-art open-source toolflows for mapping CNNs to FPGAs, focusing on fpgaConvNet. Finally, we review research into the implementation of dynamic DNNs on FPGAs, providing context for our toolflow.

2.1 Input-dependent Computation in DNNs

As has been highlighted in the introduction, the aim of input-dependent computation in DNNs is to reduce the average computational workload with minimal degradation in accuracy. The dynamic behaviour customises the computational workload per input, allowing the trade-off between accuracy and computation effort to be explored. This is a broad aim that many new methodologies now work towards, as illustrated in Figure 2.1. The first level of classification of these works highlights the granularity of the dependence introduced into the networks. We focus on single-image/sample-level dependence in contrast to the temporal or spatial dependence

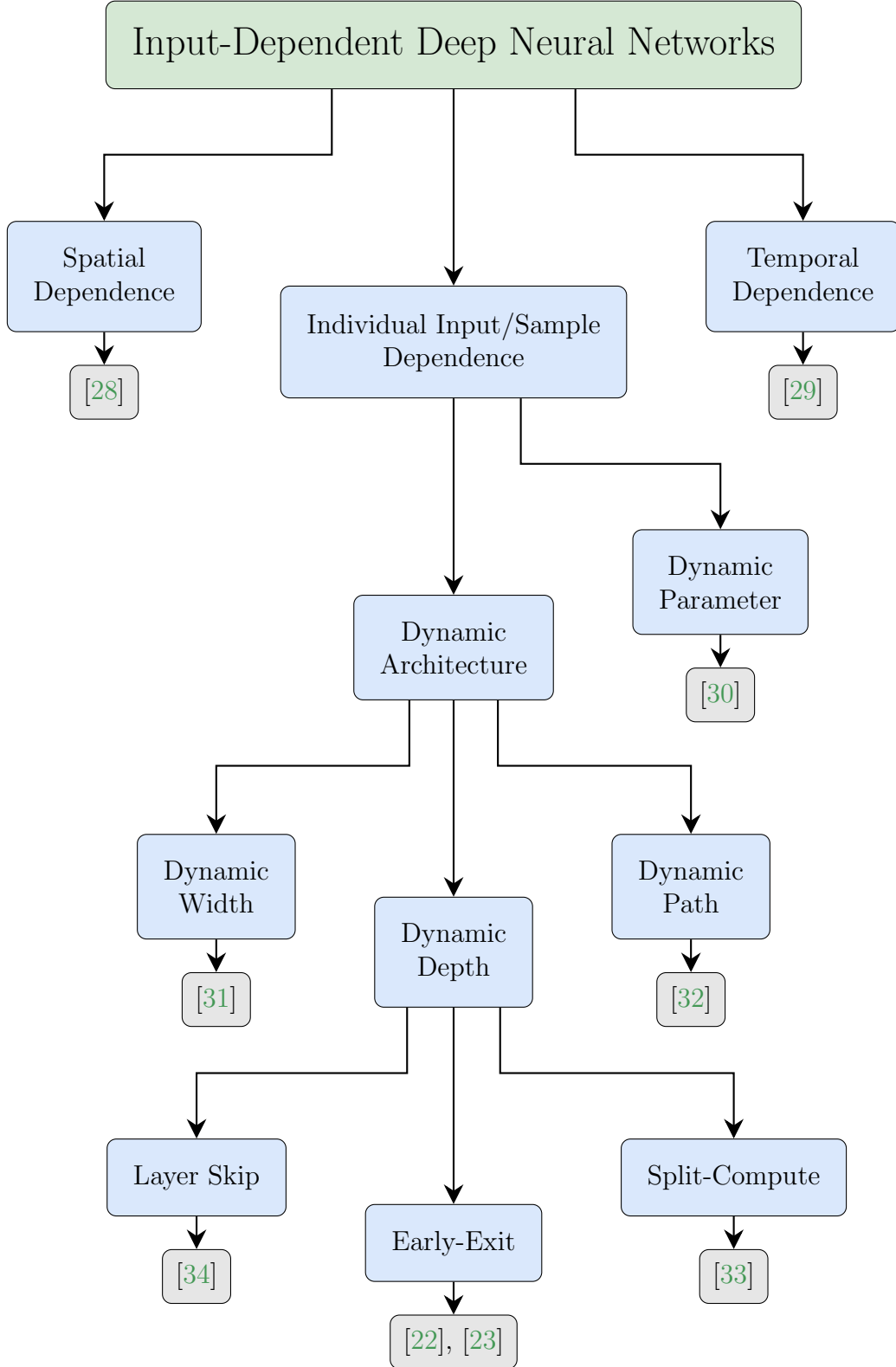


Figure 2.1: Simplified taxonomy of Dynamic Deep Neural Networks [35]. This highlights where Early-Exit networks fit into the input-dependent DNN space.

exemplified in [29] and [28] respectively or combined in [36]. With this single, input-level dependence in mind, *how* are typical static networks altered? Some works directly adapt the parameters of the network to a given input. For example, using an attention mechanism, it is possible to adjust the weight parameters on a per-input basis or apply scaled versions of spatial features within a network [30]. These methods benefit from inheriting their main structure from static networks, improving network capacity without relying heavily on special architectures. However, their training procedure is often complex and a computation overhead is introduced as the parameters are selectively adjusted. Instead, we focus on the class of dynamic DNNs that perform run-time modifications of the network architecture to reduce computation. These approaches can be further divided by the specific modifications made to the network structure at run-time.

The generalised form of these approaches can be thought of as the fragmentation of a static network into sub-networks where the path through this collection of sub-networks is determined for each input. This is exemplified in D²NN [32] where the sub-networks are ‘regular’ (convolution and fully connected) layers interspersed with ‘control’ nodes that dictate the path a given input sample will take. The full network is capable of forming any Directed Acyclic Graph (DAG), allowing a network architect to embed task information into the shape, such as cascades and hierarchies. Training is a mixture of standard backpropagation coupled with a reinforcement learning formulation to learn the control behaviour. Learning a route through a flexible architecture is challenging from a training perspective. By limiting the flexibility of the architectural changes we can improve the stability of training while still gaining the benefits of reducing average computational workload. The dynamic width approaches confine the input-dependence to channels or features. This concept is demonstrated in Dynamically Throttleable Neural Network (TNN) [31]. Their typical network architecture has network features or channels grouped into ‘T-modules’ gated by a small controller module. This controller is similarly

trained with reinforcement learning to minimise the computation of each T-module while maintaining accuracy. Lastly, we have the dynamic depth approaches. The depth can be modified by selectively applying subsequent layers as in the case of SkipNet [34]. A standard network is connected to gates that are, once again, trained using reinforcement learning to minimise the computation for a given input by dynamically controlling which layers are sequentially applied to the input sample. Unfortunately, the introduction of reinforcement learning can lead to an increase in complexity of the training procedure of these dynamic networks.

Returning to the core aim, we primarily want to reduce computational workload. Split-computing [37] methods achieve this by partitioning a network across devices capable of inherently different workloads. Computation is divided between edge device and host server: at one extreme, there is fully local computation, where inference is performed on an embedded device, and at the other extreme, data is captured and compressed locally [33] before being transmitted via a wireless connection to server for inference. In general, the local computation performs some initial classification (or other task) and assesses the confidence in the result. The local compute can then decide whether or not further computation is required and can potentially skip the high latency, round trip of transferring data to power-intensive, server-based compute [38]. A benefit of split-computing for dynamic depth is that the server-based component of the network will receive a partially computed result it only has to refine.

Having provided a details of the variety of DNNs that incorporate some level of input-dependence we now transition to a discussion on Early-Exit networks. This includes their general structure, how they make decisions on whether or not samples should exit early, and how they are trained. Early-Exit networks share structural similarities with split-computing and have drawn increasing interest in recent years [24]. We narrow our focus to this class of dynamic depth networks as we will see that their function is intuitive and they are effective at reducing average compute.

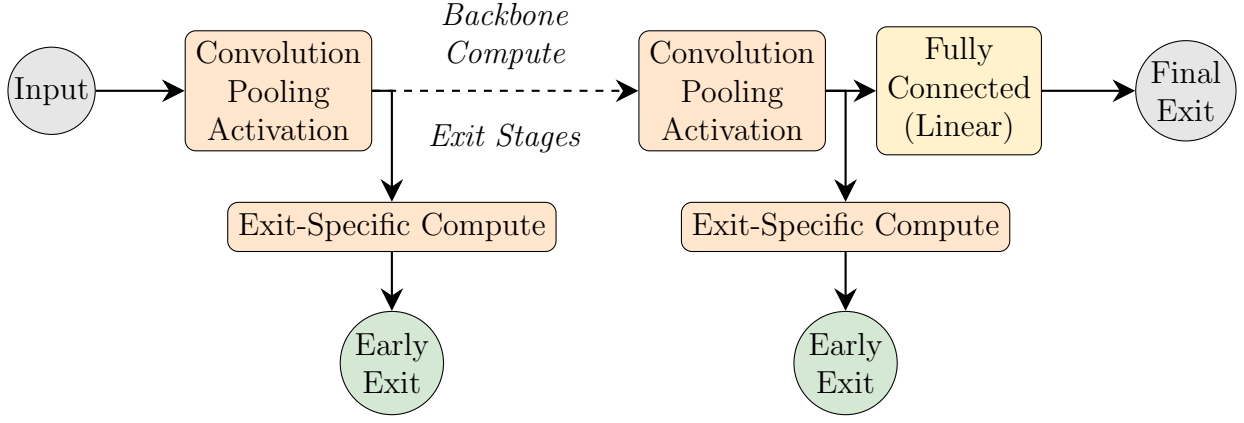


Figure 2.2: The form of a generic Early-Exit network with backbone stages and varying levels of compute between exits.

2.1.1 Early-Exit Networks

The typical architecture of an Early-Exit network is illustrated in Figure 2.2. It consists of a *backbone*, where the majority of the computation is carried out, and *exits*, that contain additional layers for CNN computation and confidence calculations. Some applications of Early-Exit branches in training, have been used to address vanishing gradient issues with deeper networks [39]. Due to the hierarchical nature of CNNs [40], the earlier layers in the backbone will have learnt more general or coarser features. During inference, ‘easy’ samples can be classified based on these features to an acceptable level of accuracy and more complex samples can be further processed in later stages of the backbone before final classification. We primarily focus on the benefits of computation reduction during inference.

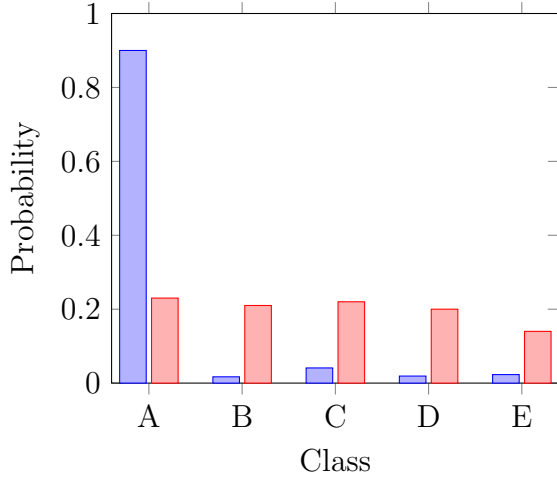
An Early-Exit network can be thought of as a collection of networks of varying depths. One benefit of using Early-Exit instead of a network collection is the shared nature of the backbone [41]. This means the additional parameters required by the network include only those within the Early-Exit branch. It also means that the intermediate results generated by the backbone sections of the network do not need to be recalculated as the input propagates through the network, reducing redundant computation. As each intermediate result reaches a potential Early-Exit point, the decision to terminate the computation needs to be evaluated with a confidence metric.

Decision Confidence

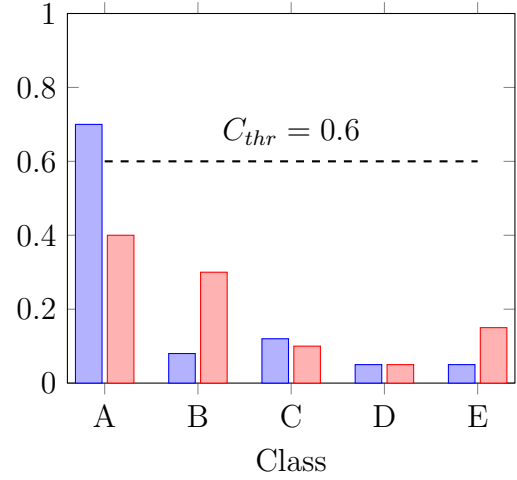
One advantage of using a confidence metric in place of a learned decision is that the threshold can be determined without additional training. However, the ‘difficulty’ of a given input is challenging to quantify so there exists a range of metrics used in the literature to judge the confidence of a result at a given Early-Exit point. For multi-class tasks which produce a vector of values associated with the possible classes, i.e., classification, the Softmax activation function is used to transform the output vector into a probability distribution over the classes. The intermediate result (the output vector x) is fed through the Softmax function: $\text{Softmax}(x)_i = \exp(x_i) / \sum_{j=1}^C \exp(x_j)$. This returns a vector of class values between 0–1 (for C classes), acting as a probability distribution for the confidence calculation.

The most common method of determining confidence is to measure the information entropy over the available classes [22], [24]. In this case, a suitable threshold is chosen, often with reference to some validation data set. We visualise high and low entropy distributions in Figure 2.3.1. A high entropy value is characterised by a more uniform distribution of probability across the Softmax result, whereas low entropy distributions have a higher concentration of probability at the predicted class value. When the calculated entropy value is lower than the threshold, this indicates more certainty in the chosen class. A slightly simpler method is to look solely at the maximum value of this probability distribution and determine if it is above a different threshold. This Maximum (Top1) Softmax metric is depicted in Figure 2.3.2. We see this method used to assess the confidence level of intermediate results in [42].

We note that fast and accurate implementations of entropy calculations are resource intensive so the alternative, more FPGA resource-friendly methods should be supported instead. This allows us to achieve the desired effect of input-dependent computation without significantly increasing resource overhead. Motivated by the FPGA target hardware, we opt for the maximum Softmax thresholding approach. In either case, the Softmax operation needs to be



(2.3.1) Example of a low entropy Softmax distribution in blue and higher entropy distribution in red.



(2.3.2) Example of high (blue) and low (red) confidence Softmax distributions according to a threshold (C_{thr}) of 0.6.

Figure 2.3: Example of Entropy and Max (Top1) Softmax confidence metrics. In each case ‘A’ is the chosen class but with differing levels of confidence.

performed on the intermediate results. We go into more detail on our choice for the decision metric and its implementation in Chapter 4. We note the cost of implementing this metric in terms of latency and resources. To reduce this cost there has been work carried out to create alternative methods of determining confidence [43] as well as more efficient implementations of the Softmax layer [44]. We build on this work in Chapter 7 and provide a solution to some of the limitations.

The *ad hoc* nature of the confidence metric is an advantage when compared to a learned decision because it does not require complex additional training, assuming you have the Early-Exits in place. However, DNNs often exhibit ‘overconfidence’ [45] when faced with data outside of the training distribution. This means that inputs may produce much higher confidence results than expected. To mitigate this, there now exist implementations of Early-Exit networks that employ a learned confidence metric [46], significantly improving accuracy at the cost of increased complexity.

Owing to the shared nature of the backbone parameters and the variety of confidence metrics, the network architecture and training procedure can have a significant impact on the

effectiveness of an Early-Exit branch. While we do not directly contribute directly to the body of novel research on architecture and training, we provide the open-source repository, [Early-Exit Net](#), created to reproduce Early-Exit networks from the literature used in the evaluation of ATHEENA.

Architecture & Training

The earlier implementations of Early-Exits make minimal changes to the backbone network architecture and have a simple two-stage training procedure in which the backbone is pre-trained, followed by the joint training of the backbone with the Early-Exit classifiers [22]. The architectural differences between these models are a result of the hand placement of Early-Exit branches and the additional convolution and linear layers to perform down-sampling and feature extraction at the different exit points. The disadvantage of the joint training method is the potential for the gradients, calculated through the backpropagation of a weighted summation of the loss of the different Early-Exits, to interfere with each other. This can prevent convergence and impact the accuracy of the backbone network. The pre-training of the backbone limits this effect, conversely, other works freeze the backbone weights when training the Early-Exits [47], [48]. The aim of the latter approach is to reduce the cost of training Early-Exit networks for multiple devices. The approach starts with having a well-trained backbone to which Early-Exit classifiers are attached. The exit parameters are then trained alone. While this limits the accuracy of early exits, it means that there is no gradient interference with the backbone. In the case of [47], these additional exits are attached and trained at every possible point on the backbone network. The exits are then assessed at design-time and only those that meet accuracy requirements are implemented for inference. In addition, the exit computation is profiled so that the network can be more closely tailored to the target hardware. We see in [49] that this idea of customisation is a promising direction with Early-Exit networks. It allows

a core model, augmented by Early-Exit, classifiers to customise the network behaviour to a specific user’s data distribution in a semi-supervised setting.

Alternatively, Multi-Scale Dense Network (MSDNet) [42] was constructed with horizontal feature extraction built into the overall network instead of merely attaching exits to an existing CNN backbone. This introduces dense connections across the convolution layers, improving the accuracy of Early-Exit classifiers by maintaining as much useful information as possible in the network. Unfortunately, the training cost is increased because all parameters have to be jointly trained from scratch using the weighted sum of the loss at each exit classifier.

There has been work carried out to improve the training methods of Early-Exit networks using knowledge distillation techniques [50], [51]. These works make use of the classification probabilities generated by the final exit to inform the learning of the earlier exits through an additional weighted term in the loss function of the early exits. This is shown to improve the accuracy of the networks as well as reducing the average workload by causing more samples to exit early. In addition to knowledge distillation, [50] scales the loss of each exit to reduce the variance of the gradient in the overlapping sections of the network to improve convergence and reduce training time. Having explained the core concepts of Early-Exit network architectures, their training and evaluation of confidence, we now look at how some of these features are used in different tools.

Our work focuses on established Early-Exit networks that consist of trained backbones and exit stages. Recent developments in this space introduce significant shifts in the Early-Exit architecture. [52] employs NAS to address the placement of Early-Exit classifiers, factoring in the accuracy and the amount of computation required at each stage of the backbone. This work estimates the accuracy and calculates the computation of a range of Early-Exit networks, employing a genetic algorithm to evolve the best networks, producing networks with well-refined Early-Exit branch placements. In all of these Early-Exit networks there is an aspect of uncer-

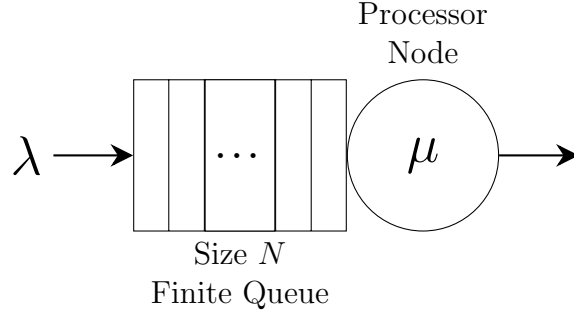


Figure 2.4: Simple queueing system with a single queue of finite size N , an arrival rate of λ , and service/departure rate μ .

tainty when encountering the run-time distribution of inputs with variable difficulty. For the majority of the networks with early exits deployed on typical hardware (CPU/GPUs), as long as it is possible to reduce the average computation enough to offset the exit branch overhead, there is a positive case for the addition of dynamic behaviour to standard networks. However, when trying to optimise the placement of exits with respect to the target hardware (as in [47], [52], and our work), it is important to be able to quantify the performance uncertainty and the impact on the Early-Exit network and target device because this directly affects the potential throughput gains. In Chapter 4, our initial implementation of the ATHEENA toolflow relies on a fixed estimation of probability coupled with resource assumptions to estimate the accelerator performance. However, our work detailed in Chapter 5, builds on this by incorporating a better estimation of the probabilistic behaviour of our full accelerator system. This updated modelling approach is based on the abstraction of the Early-Exit accelerator hardware being treated as separate processing nodes linked by a finite queue. Queueing Theory provides methods for the probabilistic modelling of distributed systems so can naturally be applied to our accelerator designs. This quantifies the performance uncertainty as well as generating throughput predictions under finite queue (buffer) constraints.

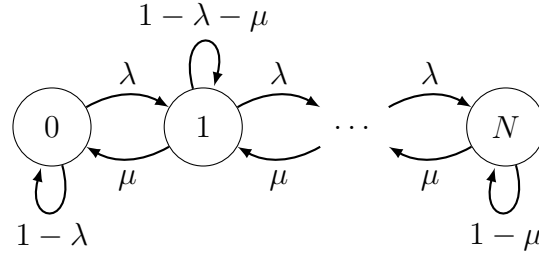


Figure 2.5: Markov chain representation of a discrete-time finite queueing system with capacity to store N items. λ is the probability the queue increments by one and μ is the probability that the queue decrements by one.

2.2 Modelling Probabilistic Systems: Queueing Theory

A typical queueing system is modelled as: a series of tasks that arrive with random intervals that follow some distribution; an arrangement of processing nodes that complete provided tasks in a time dictated by a further distribution; all interleaved with queues that temporarily hold a task ahead of downstream processing nodes [53]. Within the domain of Queueing Theory, these systems (or simplifications thereof) are described using Kendall Notation [54]. This characterises queueing systems in terms of Arrival Process (A), Service Time Distribution (S), and Number of Servers (N_S) in the form: A/S/ N_S . The Arrival Process dictates the distribution of time periods between each new arrival into the queue. Service Time Distribution is the set of time periods that it takes to process a member of the queue. Number of Servers equates to the number of processors that can operate concurrently. Figure 2.4 shows a simple queueing system where the arrival time distribution is represented by a constant value λ and the service time distribution is represented by μ .

Through simulation and/or analytical methods, a systems designer is able to understand the probabilistic behaviour of tasks flowing through a system [55]. Due to the constraints of the target hardware, we are particularly interested in modelling *finite* queueing systems. The typical approach often involves representing the systems as a Markov chain, as shown in Figure 2.5. The key property the system must satisfy is the ‘memory-less’ property. A standard assumption in the field is that the process distributions are Poisson so can be characterised by

a single value dictating the average rate of arrivals or departures into the system. This satisfies the memory-less property as knowing the time the system has been active does not affect the likelihood of the next arrival into the system. A key metric in determining the stability of systems with finite queues is the traffic intensity of a given queue and node. This is defined as $\rho = \lambda/\mu$ and is the ratio of items entering the queue and leaving the queue. ρ values greater than one mean that the queue is more likely to be filled than not. In lossy systems, this results in dropping new items but in the blocking case, the upstream nodes are stalls. A traffic intensity lower than one means the system is much less likely to stall.

This modelling system can be applied to data-flow circuit design to optimise different branches by modelling the throughput [56]. In this work, each branch stemming from a condition is modelled as an $M/M/1$ system. This means that the Arrival Process and Service Time follow a Poisson distribution. The method uses the *probabilities* of navigating down a particular branch in the data-flow circuit as the average rate. Along with the initiation interval of each branch, a simple set of equations can be used to determine the throughput of the design with sufficient buffering, operating in a steady state. By adjusting the initiation intervals of the components within the branches they increase overall circuit throughput $2 - 3\times$, in a similar way to the ATHEENA toolflow. The other variables in Queueing Theory are the Queue Capacity (K), Population (P), and Queue Discipline (D) in the form $K/P/D$. A key challenge in modelling the benefits of input-dependent computation, under the various possible parameter choices, is the difficulty of expressing the throughput with respect to the probabilistic nature of the accelerator. Therefore the $K/P/D$ parameters are generally assumed to be infinite, infinite and First Come First Serve respectively as it simplifies the mathematics of the analytical approaches to computing the values representative of queueing systems [56]. However, assuming sufficient (or infinite) queue capacity will not result in optimal sizes for our intermediate buffering stages. As such, in Chapter 5 we are no longer reliant on this assumption, instead

moving to a model with finite queue sizes.

The control and data-flow architecture we use lends itself to being modelled using a deterministic service time representation since the expected behaviour is that the initiation interval of the network is constant [57]. Unfortunately, the upstream blocking [58] of the processes, resulting from hardware stalls at run-time, is not factored in. For our purposes, we can apply abstractions to the accelerator hardware in order to simplify the modelling process, capturing the dynamic behaviour of our accelerator in a reasonable time-frame. In particular, we focus on representing the probabilistic behaviour with a general distribution of service times, characterised by measures of the mean and variance. This enables us to accurately estimate the throughput by using the two moment approximation methods outlined in [59].

The preceding sections provided a primer on Queueing Theory and how it can be applied in the case of our multi-stage Early-Exit network accelerator. Part of the challenge of modelling our accelerator comes from the flexibility of the target device. To provide the relevant background on these devices, we now define some of the key characteristics of the FPGA, in order to explain the advantages and disadvantages for our application.

2.3 FPGAs

Field-Programmable Gate Arrays are flexible devices, capable of implementing digital hardware circuits from a description of the required behaviour; formalised in a Hardware Description Language (HDL) [60]; and increasingly produced using High-Level Synthesis (HLS). In contrast to a CPU with fixed hardware controlled by an instruction set, the fundamental blocks of an FPGA lie on a routing network that can be configured according to the circuit requirements. The exact operation and arrangement of the fundamental resource blocks depend on the FPGA vendor but the programmable logic section of an FPGA will consist of specific resource blocks. When designing hardware for an FPGA, the behaviour is described using an HDL or HLS,

before undergoing synthesis in which this description is converted to operations that can be arranged and mapped to the fundamental FPGA resources. The synthesised circuit undergoes a heuristically guided, optimisation process in which it is placed and routed, tying the circuit components to the physical resources and engaging the relevant connections across the routing network. Much of this is handled by tooling in an attempt to simplify programming for the user.

In addition to the routing network and IO connections, the main blocks are custom logic units containing Boolean Look-Up Tables (LUTs), able to implement arbitrary N-input boolean functions, with built-in multiplexers and registers. The on-chip SRAM blocks, typically referred to as Block RAMs (BRAMs), can store a number of bits of information at configurable word-lengths. The DSP blocks are hardened compute units optimised for arithmetic operations such as multiplication and addition. The DSP blocks are vital to the efficiency and success of many implementations of DNNs on FPGAs as they are able to increase density of multiply-accumulate (MAC) operations, compared to the custom logic units.

Many of the advantages of the FPGA are tied to the reconfigurability of the routing network and resources. For our work, this allows us to customise the resource allocation to better provision stages of the accelerator in need, as well as implement custom precision operations that may not be available on general purpose hardware. Open-source and industrial tools exist to simplify much of the process of mapping digital hardware designs, and in recent years we have seen an increase in higher-level toolflows that help map machine learning workloads to FPGAs [61], [62].

2.4 DNN compilers for FPGAs

Increasingly, the task of executing DNNs on different devices has become simplified and optimised through the introduction of Deep Learning Compilers. These frameworks can take a

high-level DNN representation from Tensorflow, PyTorch, etc., and derive a computation graph. This graph can be optimised at the network level with methods like simplification or operator fusion. The compiler can then take this and produce a low-level intermediate representation in order to undergo further optimisation, factoring in the target device characteristics and performing hardware-specific optimisations [61]. While many of these compilers target CPUs and GPUs, fewer target FPGAs due to their complexity and lack of freely available, optimised kernels. In spite of these challenges, there are a number of FPGA-specific toolflows and frameworks that can efficiently map DNNs workloads. More recent works address the issue of scalability that comes with the use of transformer models by requiring the use of multi-FPGA systems [63] for modern LLMs [64].

We focus on the acceleration of CNNs that can fit on single FPGAs. There is a large body of work concerning the automated acceleration of CNNs using custom compilers detailed in [62]. These broadly fit into two categories: single computation engine architectures and data-flow streaming architectures. The single-engines [65] typically consists of a fixed architecture on which the CNN layers are mapped, loaded, and executed in a sequential fashion. The CNN is compiled into a list of instructions. This execution can be controlled by software using an onboard CPU, or specialised control hardware incorporating finite state machines. The benefit of the single-engine approach is that it is amenable to different CNN workloads. However, there is significant benefit, in terms of resource savings and throughput increases, to customising a given architecture to the CNN workload [66]. Streaming architectures take customisation further by making use of the data-flow paradigm to produce deeply pipelined designs with specialised layers for modern CNNs [25], [67]. In addition to these works, researchers have combined the single-engine and streaming accelerator approaches to make extensive use of compact convolution operations [68] for modern CNN architectures. Standard hardware implementations of convolution may suffer from inefficiencies when depth or point-wise convo-

lutions are mapped to them, limiting the benefits of more resource-efficient CNNs. While a useful direction for future research, the majority of Early-Exit CNNs do not yet rely on these operations. Therefore, we adopt the streaming architecture method so that we can benefit from CNN-specific customisation and implement the input-dependent compute *spatially*, targeting high throughput.

2.4.1 fpgaConvNet

fpgaConvNet [25] is an automated CNN-to-FPGA toolflow able to handle common CNN layers and forms the basis for our toolflow. The toolflow produces a streaming architecture for the accelerator designs by modelling the CNN computation as a Synchronous Data-Flow Graph (SDFG) [69]. Using SDF terminology, the nodes are the computation operations such as convolution and the arcs are streams of data. This means that the template hardware blocks (compiled using Vivado HLS [70]) can perform these operations with a static schedule as well as compensate for differing data consumption rates determined at design-time. Streaming back-pressure is handled by a standard valid/ready streaming interface.

fpgaConvNet receives a trained CNN model in the device-agnostic ONNX (Open Neural Network eXchange) [71] representation. This representation is parsed to construct the SDFG and populate an initial hardware mapping with configurable hardware templates of layers and modules corresponding to the supported ONNX operations (such as convolution and pooling). The resulting hardware graph undergoes automated Design Space Exploration (DSE) to optimise the hardware architecture by distributing the FPGA resources using simulated annealing to select possible incremental transformations to the hardware blocks. It takes as input empirically-based resource models of the hardware blocks as well as the target device resource specification and the model of the mapped CNN workload. This allows an optimised design to be found within a large search space in a relatively short time. Finally, the C++ code is

generated for Vivado HLS to compile.

For larger CNNs, folding is applied on a per-layer basis to adjust the amount of parallel computation used to generate each layer output. When the input is folded, the hardware operation is performed on a subsection of the input. This reduces resource consumption at the cost of increased latency. Folding is used at two scales: coarse-grain folding at the input and output of layers and fine-grain folding of the sliding windows within the convolution module. `fpgaConvNet` was chosen as a basis for our work because of the existing hardware templates for standard CNN layers and the tooling infrastructure that provides DSE to generate a network layer configuration for hardware implementation. `fpgaConvNet` HLS is open-source and demonstrably extensible [72]–[74] in a range of contexts. In this work, we build `fpgaConvNet` into ATHEENA by extending the scope of supported CNN architectures as well as creating new hardware templates to efficiently support Early-Exit networks. As with existing comparable toolflows [67], [75], [76], `fpgaConvNet` does not support the coarse granularity of input-dependent computation of Early-Exit networks. However, works now exist that build upon some of these toolflows with dynamic acceleration capabilities that we shall detail in Section 2.5.

For ATHEENA we expand the data-flow model of `fpgaConvNet` to include input-dependent computation as pipelined control flow. This allows ATHEENA to generate CNN-hardware mappings that benefit from varying input sample difficulty. The method of extension we develop for `fpgaConvNet` is orthogonal to extreme quantisation [76] and exploitation of sparsity [67] detailed in other works. Furthermore, we improve overall compilation times by partitioning the design prior to HLS compilation and then automatically stitching the generated components before synthesis and implementation. Having outlined the range of existing toolflows that map generic DNNs to FPGAs and the open-source tool we build upon, we now detail more closely related works. Broadly, these combine a level of dynamic parameter reduction with acceleration implemented on FPGAs.

2.5 Dynamic DNNs on FPGAs

CascadeCNN [77], [78] implements a specialised form of dynamic computation where a run-time decision is made to switch between versions of the same network quantised to a high and low-precision. The components of the architecture include both a low and high precision implementation of the network on FPGA fabric and the confidence evaluation unit running on a CPU. All input samples are first fed through the low precision network before the evaluation of a confidence metric on the results to determine whether or not they meet the user-defined accuracy threshold. Any samples with low confidence are then fed through the high precision network. ATHEENA is similar, in that all input sample processing is confined to the FPGA fabric to avoid prohibitive reconfiguration costs [78]. However, we also choose to include the confidence evaluation on chip for ATHEENA, meaning data does not need to make the round trip to and from off-chip memory to calculate confidence. The key difference for ATHEENA is the streaming architecture that can achieve a higher throughput thanks to the deeply pipelined layers.

DynExit [79] uses a hand crafted, single engine architecture to implement a Residual Network (ResNet) with classifiers attached along a pre-trained backbone. The network architecture consists of pipelined convolution and linear (fully connected) execution blocks attached to a ‘branch’. The ‘branch’ consists of an exponential and a natural logarithm module to compute the confidence of the classification based on a rearranged version of the cross entropy loss function. ATHEENA also uses a dedicated block to determine the sample confidence. However, DynExit’s lack of customisable layer configurations per network limits the ability of the design to fully utilise the FPGAs resources.

Adaptive Hierarchical CNN (AHCNN) [80] notes the benefits of being able to utilise the shallow features for ‘easy’ input sample classification and deeper features of a more accurate network for ‘difficult’ classifications. To this end, partial reconfiguration is used to swap in and

out shallow and deeper sections of a ResNet18 CNN. Large batch sizes are used to amortise the latency penalty for reconfiguration. This allows the design to benefit from full utilisation of the resources for each stage of network computation but the latency penalty is prohibitive for low latency applications. They also compute the confidence decision based on the maximum value of the Softmax function applied on the network output, as with ATHEENA. However, this is combined with the profiled probability of the occurrence of a given class, resulting in a confidence threshold that is class-dependent. The hardware architecture for AHCNN consists of fixed convolution and pooling layers ahead of common fully connected layers and a decision gate. The result from the gate determines which bitstream should be loaded via partial reconfiguration on the FPGA fabric. This allows the design to benefit from full utilisation of the resources for each stage of network computation but the latency penalty is prohibitive for low latency applications.

With a similar procedure to DynExit, the heterogeneous architecture proposed in [81] makes use of the FPGA fabric to accelerate the highly parallel CNN computation through the use of multiple processing elements in a systolic array architecture. This single-engine can be tuned to a given device but still requires the confidence decision to be calculated by the CPU. The onboard CPU computes the Softmax function, followed by calculating the entropy of the intermediate classification results. As with DynExit, this is another hand-crafted architecture that supports the main convolution kernel sizes but currently has limited support for more recent networks.

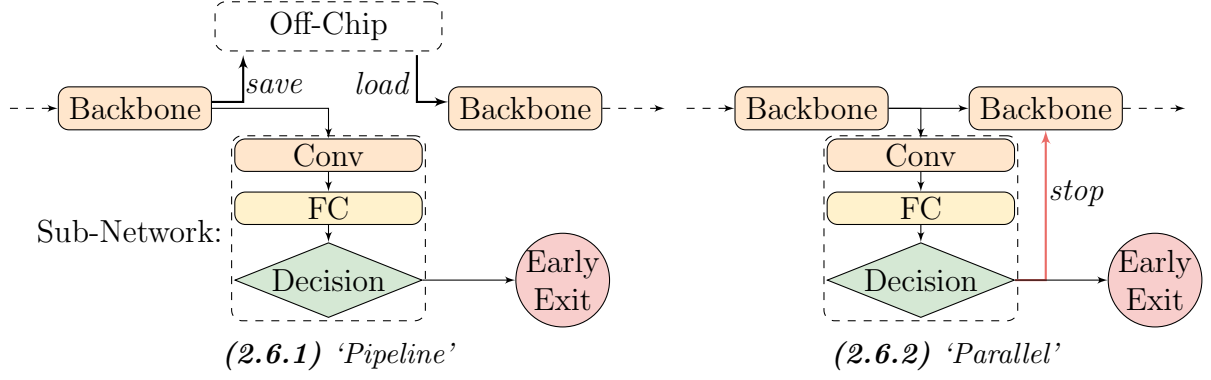


Figure 2.6: Working principle for the different approaches to Decision Sub-Networks [82].

The work proposed in [82] details two hand-crafted hardware architectures for accelerating the inference of Early-Exit networks on FPGAs. The pipelined approach, depicted in Figure 2.6.1, adopts a traditional Early-Exit network topology. The inference procedure is as follows: an input is processed by stages of the backbone in a pipelined fashion until reaching the layer preceding the decision sub-network. The processing of the backbone is stalled as the intermediate results are transferred off-chip until the results of the intermediate classification are computed and an Early-Exit decision is made.

This architecture allows for the reuse of existing hardware modules, achieving nearly 0% additional area overhead. While this may result in some under-utilisation, the overhead reductions is often required when targeting the low resource setting of FPGA acceleration.

The disadvantage of the pipeline approach is the significant latency introduced from the intermediate results being transferred on and off of the FPGA fabric. Conversely, the parallel design approach, highlighted in Figure 2.6.2, enables the full parallel processing capabilities of the FPGAs to be exploited. The Early-Exit branch is executed in parallel with the backbone network, utilising separate hardware modules. This means the intermediate result is fed into both the next layer of the backbone network and the Early-Exit branch. This approach mitigates some of the latency overhead during Early-Exit branch computation in addition to preventing the storage of intermediate layer outputs. The limitations of this implementation lie in the high resource consumption for the additional computation and the potentially wasted

compute as the intermediate results are allowed to propagate down the later stages of the backbone.

Though our work was one of the earliest to produce a toolflow that can take Early-Exit CNNs and generate accelerators for FPGA devices, there now exist many related toolflows for generating and managing accelerators for the execution of dynamic DNNs. These are summarised in Table 2.1.

Table 2.1: *Summary of the reviewed dynamic toolflows.*

Toolflow	Accelerator Class	Network Architectures	Target Devices	Early-Exit Support	Year
AdaFlow [83]	Streaming Dataflow	FINN-based, Pruned BNNs	Zynq Ultrascale+	No	2022
DPACS [84]	Streaming Dataflow	Custom ResNets	Zynq Ultrascale+	No	2023
AdaPEX [85]	Streaming Dataflow	FINN-based, Pruned, EE BNNs	Zynq Ultrascale+	Yes	2023
ATHEENA [26]	Streaming Control & Dataflow	<i>General</i> EE CNNs	Xilinx Devices	Yes	2023

2.5.1 Toolflows

The AdaFlow [83] tool is built upon the FINN [76] CNN acceleration on FPGAs. AdaFlow compiles a collection of network versions pruned to increasing degrees of sparsity. The accuracy of these pruned network accelerators is inversely correlated with their sparsity, meaning that more sparse versions have lower accuracy but benefit from a significantly reduced latency. At run-time, the user can specify an accuracy threshold and the system manager selects the appropriate pruning level with a fixed or flexible switching architecture.

DPACS [84] is a toolflow implementing dynamic pruning on FPGAs. This includes specialised hardware blocks that can dynamically apply the sparsity for each input as opposed to switching between different sparse versions of the network. This work utilises spatial masks that

apply to channels per layer as well as resolution masks that can apply across residual blocks in ResNet CNNs. The limitations of DPACS are similar to that of static pruning. We can see from their results that the large-scale ResNet implementations can achieve $2 - 3\times$ speed-up. However, the speed up for MobileNetV2 is reduced to $1.14\times$, likely due to the significantly lower redundancy of the parameters of this hand-crafted network.

Building on AdaFlow, AdaPEX [85] is a related FPGA work that explores dynamic network inference on FPGAs. This tool makes use of ‘adaptive’ pruning and Early-Exit intermediate classifiers to explore the trade-off between resources and throughput at run-time. AdaPEX is, again, built on the FINN [76] HLS framework, and used to construct multiple FPGA accelerators for a FINN’s quantised DNNs. The key difference with this work is that the full system is coordinated by a run-time manager. Each of these accelerators has a different level of channel-wise pruning, with reduced accuracy corresponding to improved accelerator performance. The runtime manager is responsible for orchestrating the reconfiguration of the FPGA based on the user requirements for accuracy and throughput. Since reconfiguration can be costly, the work also provides runtime-parameterised versions of the HLS-based templated accelerators that can support multiple scales of pruning. This drastically reduces the latency cost of switching between models at the cost of additional resource overhead and some reduced efficiency. In addition to pruning, this work augments a standard DNN with intermediate Early-Exit classifiers attached along the backbone of the network. Using these additional degrees of freedom, the work can explore the trade-off between accuracy, throughput and power-efficiency. One of the similarities with this and many previous works is the use of quantisation to reduce the memory footprint of the parameters of DNNs deployed on the FPGA, as well as reducing the resources required for performing arithmetic operations. It is worth highlighting the limitations of the AdaPEX tool in relation to ATHEENA. The pool of networks supported by AdaPEX is limited to specific versions of the quantised FINN HLS library as well as to specific small datasets.

Whereas (in theory) the scope of networks supported by ATHEENA is only limited by the size of the FPGA device and the support of operations by fpgaConvNet. This dramatically increases the potential for accelerating Early-Exit CNNs.

Static *and* Dynamic Parameter Reduction

In Chapter 1, we presented input-dependent computation as a method for reducing average compute beyond the static methods of parameter reduction (quantisation, pruning, etc.). This is because we presume that the approaches are orthogonal. While not directly addressed in our research we would like to highlight some evidence in favour of this assumption. The quantisation used in [86] is 16-bit fixed-point precision, the same as used in ATHEENA. This halves the footprint, with respect to a full precision network, with no effect on the accuracy so it is reasonable to assume that the same is the case for Early-Exit classifiers trained and quantised to the same degree, as in our toolflow.

The previously mentioned AdaPEX work adds to this evidence in their use of quantised and pruned networks in conjunction with Early-Exit classifiers. Heavily pruned networks (above 45%) with an attached Early-Exit classifier exhibit a matched or higher accuracy than equivalently pruned networks with no Early-Exit classifier [85]. This result is somewhat counter-intuitive so is unlikely to generalise to larger networks and datasets. However, it does demonstrate that static and dynamic methods can be effectively utilised together. The exploration of static and dynamic parameter reduction together presents an interesting future direction the ATHEENA toolflow could be used to explore.

2.6 Summary

This chapter addressed the scope of Dynamic DNNs developed with the goal of increasing inference efficiency. The focus was narrowed to dynamic depth approaches, reviewing and comparing

these with the Early-Exit approach, and highlighting the benefits of their probabilistic nature when executing tasks with variable difficulty. The next section detailed statistical modelling of probabilistic systems using Queueing Theory, with reference to its application in improving the throughput of data-flow systems, particularly important for our accelerator. In Chapter 5, we build on this information using abstracted hardware models, rooted in Queueing Theory, after having developed a working resource allocation approach in Chapter 3. Then next section looked at target device for accelerating our DNN workloads, highlighting the principles with which existing toolflows map standard CNNs to FPGAs. Finally, we detailed the current developments in the acceleration of Dynamic DNNs on FPGAs.

While there continues to be new developments and increasing attention in the area of both Early-Exit and Dynamic DNNs in general, there is shortfall between the advancements of accompanying tools [27]. As such, a key challenge in migrating these software-oriented Early-Exit designs to FPGA hardware, is to develop a hardware architecture that efficiently executes the variety of input-dependent control-flow whilst minimising extraneous computation, area overhead, and maximising the available throughput gains. These are the primary aim and objectives 1 & 2 listed in Section 1.1 and the central challenge we first address with ATHEENA in Chapter 3 and 4. The lack of Early-Exit support in AdaFlow and DPACS as well as the limited scope of the supported networks in AdaPEX motivate the creation and improvement of the ATHEENA toolflow, detailed across the technical chapters of this dissertation.

CHAPTER

3

Method for the Efficient Acceleration of Early-Exit CNNs on FPGAs

The following chapter details the initial methodology of the ATHEENA [26] tool, published at FCCM 2023. The chapter outlines the theoretical intuition driving aspects of the Design Space Exploration and resource balancing.

3.1 Introduction

Having motivated the benefits of both Dynamic DNNs and the acceleration of Deep Learning workloads using customised accelerators on FPGAs, as well as highlighting the combination of these two concepts, our goal is create a toolflow that can improve the throughput or reduce the resource cost of the accelerator designs. In this Chapter, we address objective 2 stated in Section 1.1. The fundamental issue of implementing Early-Exit networks on FPGAs is determining a hardware mapping that balances the control and data-flow throughput requirements whilst minimising latency overhead. The baseline fpgaConvNet can optimise the hardware of a DNN by balancing the throughput across different layers. This task is complicated when the networks have data dependencies that bridge multiple layers, and network stages that may or may not be required to execute for a given input. A simplistic implementation of the Early-Exit

network, akin to the baseline, makes no use of being able to skip redundant computation on a confident classification, incurring an overhead cost for marginal latency gains for *some* inputs, and, due to the pipelined nature of the accelerator, gaining no benefits in terms of average throughput. The next rung up introduces a decision component with a control mechanism to prevent downstream stages of the network from exiting if an early result is deemed confident. With this in mind, a naive implementation would have all stages of the network optimised for the highest possible throughput. However, in the presence of any resource constraints this is clearly a wasteful strategy: having all stages targeting the same fixed throughput will lead to some stages being under-resourced bottlenecks and others being starved of data samples. Hence in this work we target the following problem: given an FPGA platform with certain computational and memory resources, what is the best way to allocate those resources to maximise throughput for a given expected distribution of samples of varying difficulty. We now define our methodology for determining this optimised resource allocation for Early-Exit networks, utilising the probabilistic nature of the accelerator under the Early-Exit task in order to inform the design choice.

3.2 Scaling Resource Allocation by Exit Probability

We exemplify the initial resource allocation process using a two-stage network in order to reduce complexity in the explanation, we briefly outline how this method can be extended for multi-stage networks in Section 3.2.1. Furthermore, our explanation looks at the generic two-stage network shown in Figure 3.1, we ground this example with a full case study and implementation of the BranchyNet in Chapter 4. We first partition our Early-Exit networks into the computational stages before each exit (including the final exit). We do this at the layer level in order to group all the components that need to operate at their common target throughput. The first stage contains all the parts of the backbone and the Early-Exit layers that are

required to operate at the higher data rate. This includes the standard CNN layers, made available by the baseline fpgaConvNet for carrying out the convolution, pooling, and ReLU functions. Each layer has a parameter configuration that enables the trade-off between parallelism (typically resulting in higher throughput) and resource consumption. Here, we briefly introduce some important additional components in order to achieve the desired function of the Early-Exit network. The goal of these components is to introduce the conditional operation of the Early-Exit networks, allowing the accelerator to reduce computation on a per-input basis. The Decision Component takes in a preliminary classification result from the exit branch and determines if the confidence of the result is above a user defined threshold. The conditional buffer will temporarily store an intermediate computation from the point at which the exit branch diverges, and only releases this partial result when the decision component has finished determining whether or not to exit. Specific details about these layers are outlined in Chapter 4.

The second stage contains the remaining parts of the backbone and final exit, made up of standard CNN layers. Due to the dynamic, input-dependent nature of the Early-Exit networks we know that not all input data samples will be required to pass through to the second stage. This is made possible by the conditional behaviour of the new components. For an accelerator in which the throughputs of the first and second stage are matched, this results in an under-utilisation of the second stage. Observing this, the second stage is only required to operate at a lower throughput. In turn, a lower throughput target means that the resource requirements of the second stages can be reduced appropriately. This reduction in utilisation of the second stage is estimated by the probability that an early exit occurs, denoted by $1 - p$. This probability is determined by the Early-Exit profiler component of the ATHEENA toolflow, detailed in Section 4.2.2. As the first and second stage are coupled together as part of a pipeline, a reduction in the use of the second stage equates to a increase in the *effective* throughput of this second stage, by a factor of $1/p$.

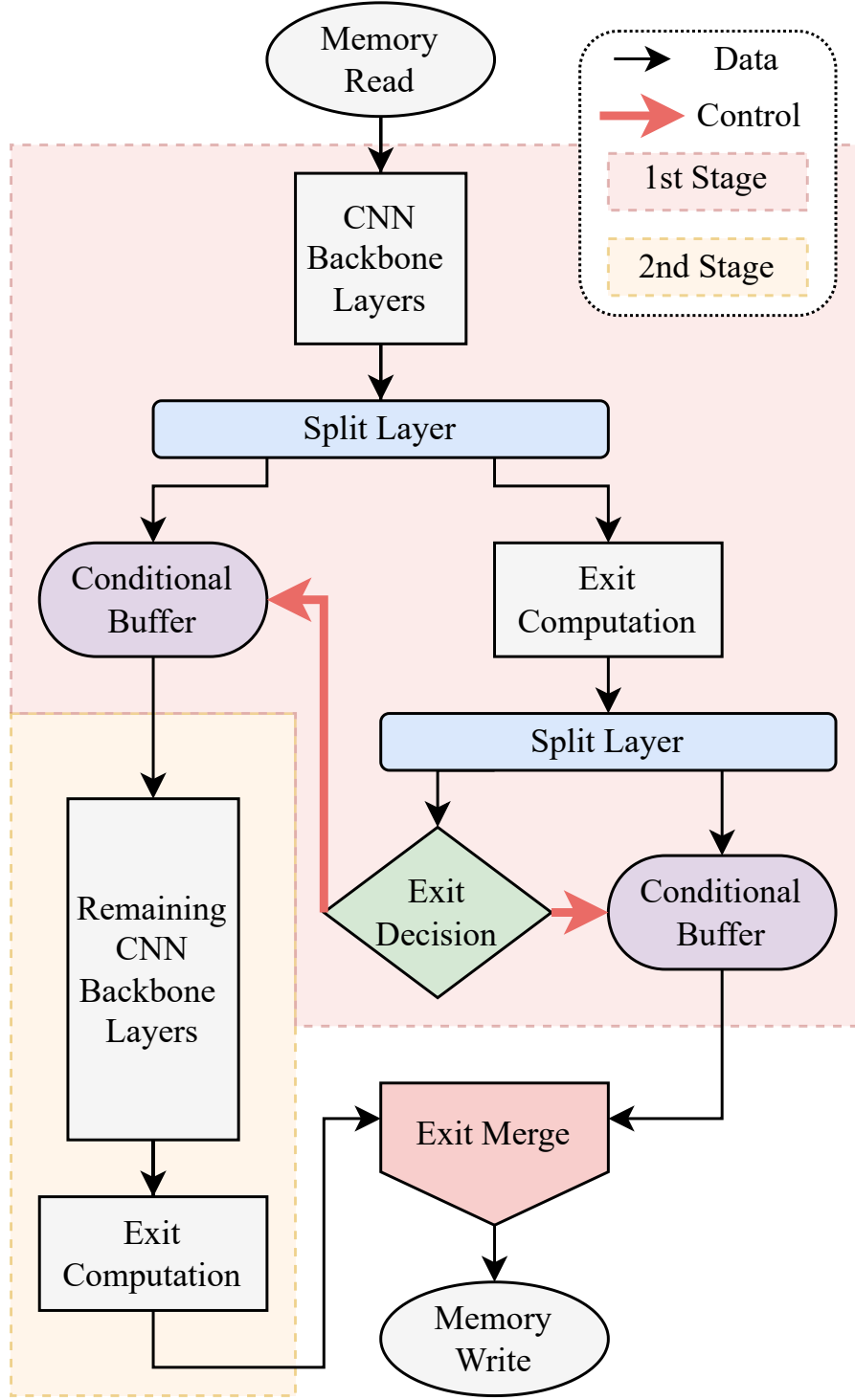


Figure 3.1: High-level diagram of the proposed control-flow hardware attached to the CNN layers of fpgaConvNet for a two stage network. Black arrows represent normal data-flow plus Sample ID tags and red bold arrows represent control signals.

The primary goal of the ATHEENA optimiser is to find the best balance of the finite available resources and design an accelerator architecture able to support the differing throughput requirements. To achieve this, we make use of existing FPGA design tools capable of folding or tiling DNN operations (in our case `fpgaConvNet`) in order to generate optimised Throughput-Area Pareto (TAP) functions for both stages. Each stage targets the same clock frequency. This means that optimisation of throughput with respect to area is no longer interdependent. Area becomes synonymous with the FPGA resources consumed. We then merge the TAP functions by combining the different stages, as depicted in Figure 3.2. This process is repeated for each combination of stages in order to generate a combined TAP function. This allows us to estimate the best use of resources under specific exit probability distributions.

We define a TAP function as a function that is (non-strictly) monotonically increasing in each of its arguments. The principle is that this function captures the maximum achievable throughput possible by separately optimising a section of the network for a given fraction of total resources. Let $\mathbb{N}$ denote the set of natural numbers and $\mathbb{Q}$ denote the set of rational numbers. An example of a TAP function might then be, $f : \mathbb{N}^4 \rightarrow \mathbb{Q}$, capturing the optimal throughput possible with a constrained number of BlockRAMs, DSPs, FFs, and LUTs, as represented by the four arguments to the function. A function like this is automatically generated by providing the `fpgaConvNet` optimiser limited fractions of the board resource constraints. The results for each set of constraints are collated for input to the ATHEENA optimiser. The *1st Stage* and *2nd Stage* graphs of Figure 3.2 show a sketch of some TAP functions for the first and (scaled) second stage of a two-stage network.

Our objective is to combine these two TAPs into a single TAP for the Early-Exit network. We may think about the problem in this way: since the second stage is only expected to be used by each input sample with some probability $p \in (0, 1]$, it follows that, given suitable buffering between stages, it is possible to extract a higher throughput than the nominal throughput of

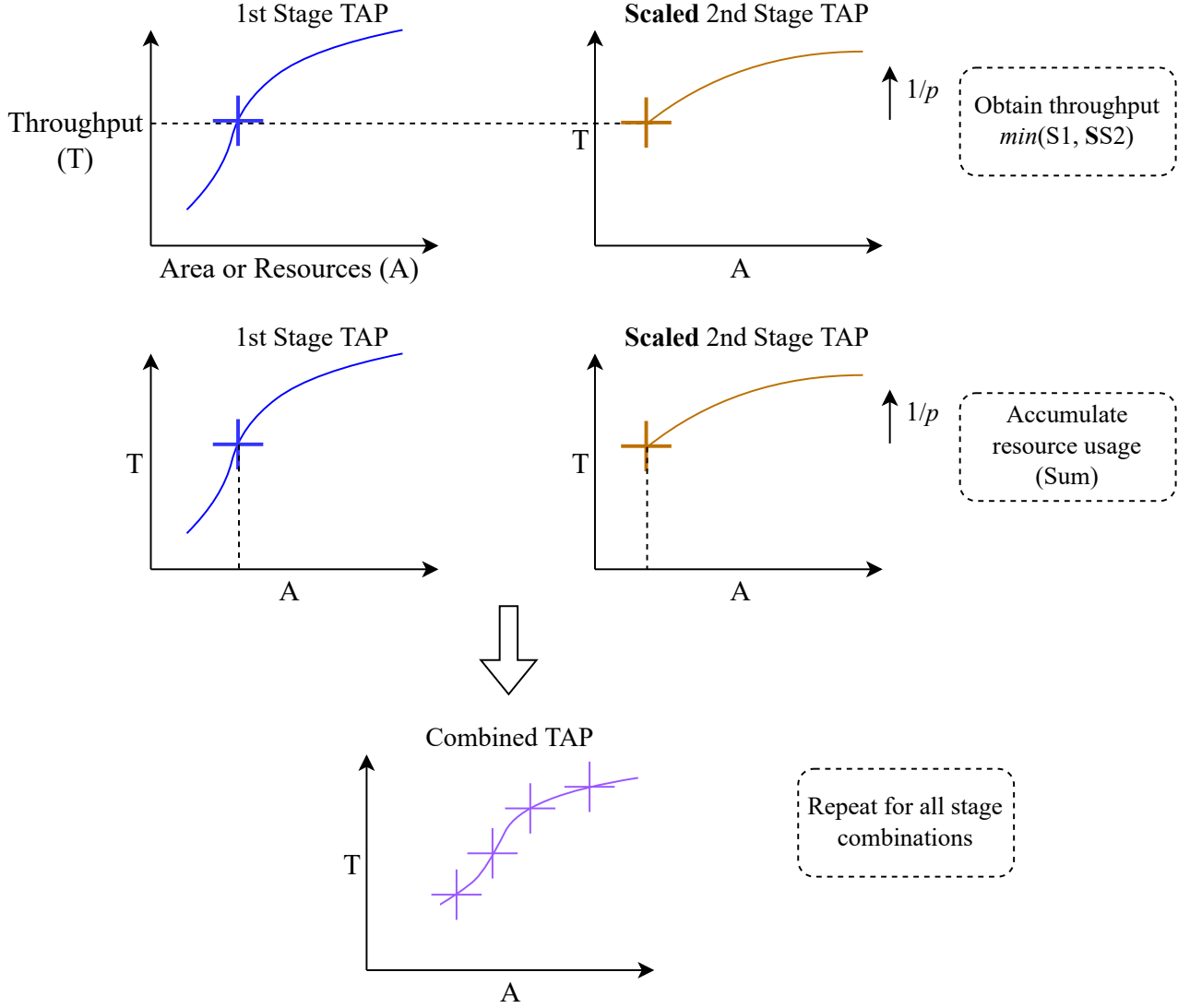


Figure 3.2: The first and second stage TAP functions shown in the top four plots are made up of discrete design points generated by calls to the *fpgaConvNet* optimiser. To generate the Combined TAP function we first select a point on both the first and second stage TAP. To obtain the throughput for this combined design point, we take the minimum (limiting) value of the throughput for the first and second stage. To obtain the resource usage, we simply sum the resources consumed by the first and second stage. This provides us with a coordinate that we can plot on the Combined TAP curve. This process is repeated for all combinations of stages and the dominant ones are saved as the combined TAP.

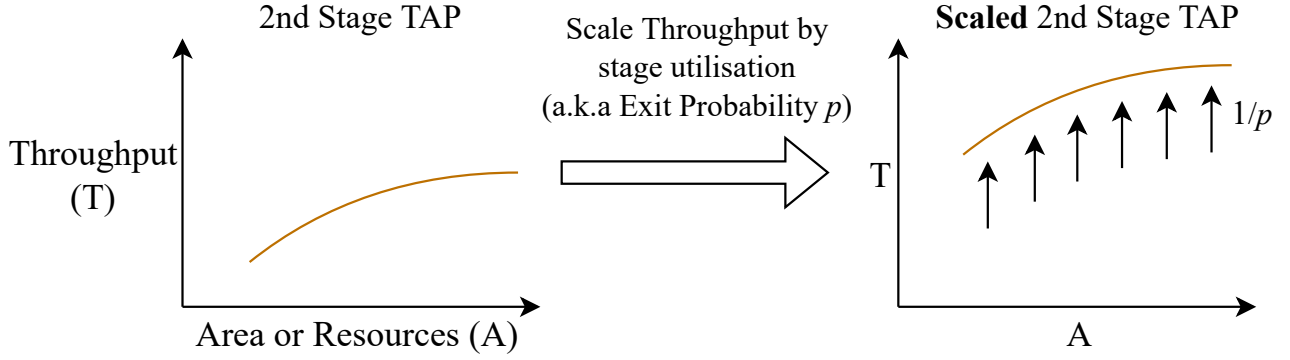


Figure 3.3: The *fpgaConvNet* optimiser produces the TAP function for the second stage in the left hand plot. Since this stage will be utilised with probability p we can scale the throughput of this stage by $1/p$ as a input only has a p chance of making it to the second stage. For example, if $p = 0.25$ (25%) then for every 4 inputs that pass through the first stage, 1 input makes it to the second stage. This means that, in the time it takes the accelerator to process these 4 inputs, the second stage has only had to process 1. Therefore, the effective throughput of the second stage is $4\times$ greater, allowing a significant reduction in the resources required by this second stage without affecting the accelerator throughput.

this second stage, by a factor $1/p$. We represent this change visually as the scaling of the throughput values of the second stage, depicted in Figure 3.3. At this stage in the development of the methodology, the assumption made is that the buffer size is infinite. In Chapter 5 we return and refine this assumption to more accurately model the throughput.

The overall design will be limited by the throughput of the first stage or the second stage scaled by $1/p$, whichever is lower. However at the runtime of the accelerator, the stochastic nature of the distribution of easy and hard samples will result in deviation, δ , from the profiled probability for which the hardware was designed, resulting in some over/under-utilization of the second stage. The profiled, design-time probability estimate is defined as p and the run-time (real-world) probability is q . Amalgamating these definitions, we are able to construct an ideal combination of the two TAPs, parameterised by p and q , expressed formally in (3.1) where $\oplus$ denotes the so-defined TAP combination operator. This operator represents the procedure shown in Figure 3.2.

$$f \oplus_{p,q} g : x \mapsto \min(f(x_1), g(x_2)/q) \quad (3.1)$$

$$\text{where } (x_1, x_2) = \underset{\substack{x_1, x_2 \\ \text{s.t. } x = x_1 + x_2}}{\arg \max} \min(f(x_1), g(x_2)/p)$$

f and g refer to the first and second stage TAP functions respectively. The $\oplus$ operation is defined as their combination, resulting in a real-world, run-time prediction of throughput for a given resource budget $(x_1 + x_2)$. Intuitively, what this equation tells us is that for a given total resource budget x , we should allocate x_1 resources for the first stage and x_2 for the second stage in order to maximise the throughput of the limiting stage, taking into account our design-time estimate of probability p . At runtime, the maximum throughput demand on the first stage will be as expected, but the utilisation reduction of the second stage may vary. This is indicated by the use of $q = p \pm \delta$ in the evaluation of the predicted accelerator throughput, instead of p used in the optimisation stage. We illustrate these differences in design-time and run-time probability in Figure 3.4 as a shaded region surrounding the main TAP curve.

In practice, the design points represented by the TAP function for the first and second stages are discrete. This means that it is unlikely for the tool to be able to perfectly match the predicted throughput values. In the case that the second stage is the limiting factor, a reduction in the use of this stage, corresponding to $q < p$, will result in an increase in throughput. For $q > p$, the throughput will be reduced due to the reliance on the limiting stage. These situations are represented by the shaded region of Figure 3.4. The solid purple line corresponds to $q = p$, in which the probability of hard samples matches that of the profiled, design-time probability. We design an Early-Exit profiler to obtain our probability profiles and use them to inform the combination of TAP functions in the manner previously described. ATHEENA builds on fpgaConvNet and automates this process for the user.

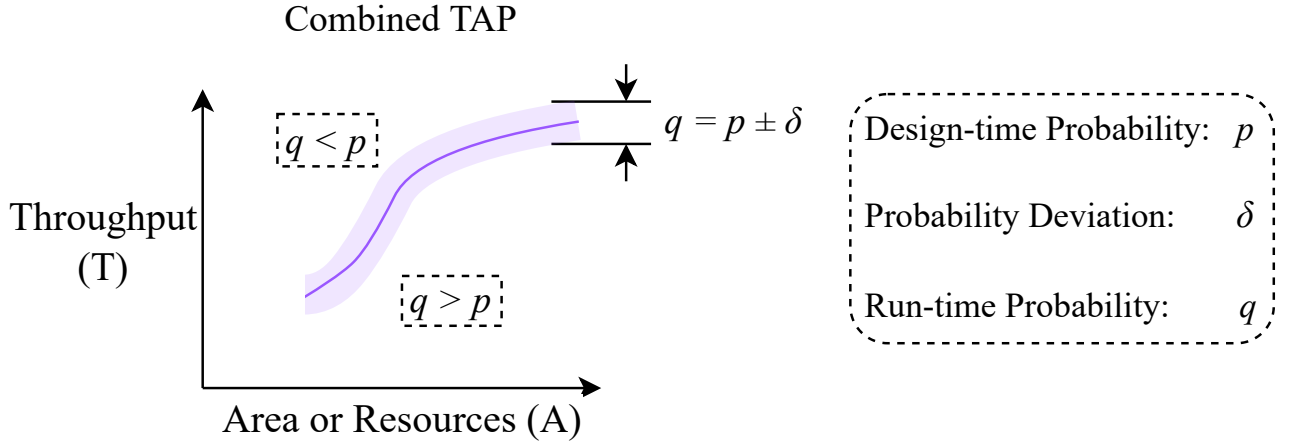


Figure 3.4: The Combined TAP function shown in Figure 3.2 depict the ideal case – where the throughputs of the first and second stage match. Additionally the run-time probability q is assumed to be identical to the design-time p . In reality, these cases may not occur consistently. As such, we provide the shaded region surrounding the TAP function. This shaded region is calculated by altering the design-time probability p by some small deviation δ . The throughput predictions for the resulting run-time probability q capture cases in which there is an increase or decrease in accelerator throughput for a given resource budget.

3.2.1 Multiple Early-Exit Stages

Early-Exit networks have been shown to be effective at reducing average computation with more than two exits [22], [42]. So, it is reasonable to ask, can our methodology be similarly applied to multiple exit stages? In short, yes. There is no reason to think that the same procedure of: fragmenting an Early-Exit network into multiple stages; constructing TAP functions for each of these stages; scaling the stages according to their predicted usage p ; and then combining these stages to find the optimal TAP function, will result in sub-optimal accelerators. However, we have not been able to carry out on-device experiments to verify this. Therefore, to supplement this discussion, we refer to cases in the literature and our own predictions for increasing the number of exit stages for a given network.

The BranchyNet work describes three networks, B-LeNet has two stages but B-AlexNet and B-ResNet need to have three [22]. This extra exit is required to maintain the accuracy of the network on the larger datasets while still reducing enough of the average computation. The exit percentage of B-LeNet is about 95% at the first exit, compared to about 60% and 40% for the larger networks. Having an additional exit stage before the final exit, reduces the

utilisation of the final stage by a further 25% and 14% respectively. With these statistics we can determine that the FPGA acceleration of the B-AlexNet network would likely result in a significant performance benefit, however, the acceleration of B-ResNet would be much reduced due to the higher proportion of input samples making it to the final stage.

The disadvantages of additional exits lie in the robustness of the accelerator. A part of the performance improvements are the result of accurately predicting the proportions of input samples that make it to each stage of the network. Variations in the run-time probability are mitigated by buffering between the stages. Increasing the number of stages increases the chance of throughput variation as well as increasing the buffering requirements. For the GPU-based acceleration case, the trade-off between accuracy, number of exits, and computation reduction is explored as an optimisation problem in HAPI [47]. The effectiveness of a given exit is determined by the reduction of computation for a single sample multiplied by the number of samples that can be removed early, given a particular accuracy requirement. This means that the optimal placement of exits and the number of them is determined by the level of training that each exit branch receives. It would be possible to extend this approach to the ATHEENA toolflow but a reliance on the on-device profiling used in HAPI would greatly increase the compilation time of the toolflow.

3.3 Summary

This chapter explained our approach to leveraging the probability distribution of task inputs with differing levels of difficulty to efficiently partition the resources of a given FPGA board in order to maximise the overall throughput. We detailed the method by which TAP functions are combined to provide a Throughput-Area Pareto function for the ATHEENA accelerator designs. We explained how this TAP can vary according to changes in real-world deviations in second stage utilisation and explained how the method could be extended for multiple exit

stages. We now demonstrate the scaling methodology in practice in Chapter 4.

CHAPTER

4

ATHEENA: A Toolflow for Hardware Early-Exit Network Automation

The following chapter details the first implementation of the ATHEENA [26] toolflow, published at FCCM 2023. In this chapter we explore the practical elements of the toolflow, followed by a demonstration of the of the toolflow’s potential on the BranchyNet [22] network.

4.1 Introduction

The ATHEENA toolflow was created by extending the open-source fpgaConvNet as illustrated in Figure 4.1. The red, italicised sections of this diagram represent the novel elements, primarily implemented to support the dynamic nature of Early-Exit networks and their accurate representation across the different sections of the toolflow. In this chapter we shall explore the motivations and requirements of these developments, and how these influence the resulting design choices. Given an Early-Exit network description and profiling data set we first need to translate this into a format that ATHEENA can interpret. This means supporting the control-flow representation of front-end frameworks. The parsing functions needed to be rewritten in order to represent these branching and conditional operations. The scaling methodology in

Chapter 3 requires an estimation of early-exit profile probabilities, we introduce an additional step into the toolflow that can estimate this using the profiling data set and network description. We redesign the optimiser to be able to represent conditional operations and factor in the exit profile when constructing the parameterised spatial streaming architecture.

The existing hardware templates are augmented to support sample identification followed by the creation of parameterised Early-Exit component templates (detailed in Section 4.3). These new components implement control-flow and softmax-based confidence calculations on the FPGA. To address some of the limitations of single-threaded compilation in Vivado/Vitis HLS, we introduce a design-time stitching procedure to allow the parallel compilation of independent hardware kernels and their combination prior to synthesis and implementation. The final part of this chapter will look at a key case-study motivating the development of this toolflow. We address the behaviour of the optimised hardware in relation to our initial methodology and evaluate the benefits and limitations of the first iteration of the toolflow. This contributes to the primary aim and objective 1. from Section 1.1.

4.2 Front-end Representation

The first upgrade introduced by ATHEENA is that the toolflow is now capable of leveraging the dynamic nature of Early-Exit networks. This means supporting the use of control-flow representation in front-end frameworks like PyTorch and the device-agnostic representation of ONNX. Baseline open-source fpgaConvNet directly maps a CNN in the ONNX format to an internal graph representation in order to model the hardware as a data-flow graph. ATHEENA generates a *control* and data-flow graph (CDFG) to represent the progression of input data across layers and the confidence-based control decisions gating this data flow. This graph is used to represent the parameterised spatial streaming architecture so that we can explore the possible design configurations. We improve the parser and optimiser with modifications to support the

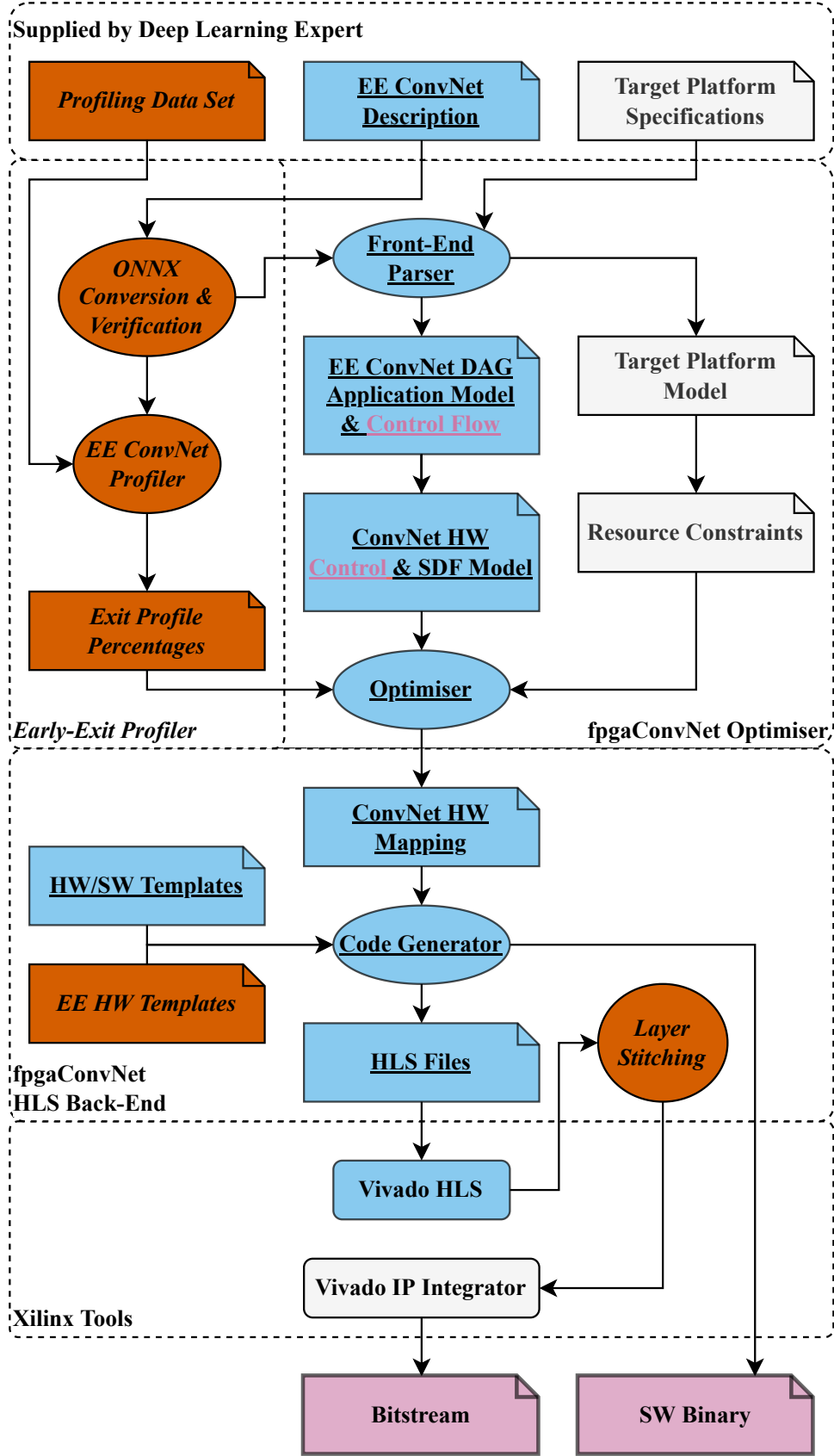


Figure 4.1: A visual representation of the ATHEENA toolflow. We have built upon the base-line, open source fpgaConvNet toolflow. New elements are in red italics and modified elements are in blue, underlined.

new required ONNX operations and mapping of control-flow for hardware translation.

4.2.1 ONNX Conversion

A notable challenge in the acceleration of Early-Exit networks is locating a source of trained networks that we can use as input to the ATHEENA optimiser. This is due to two main factors: a diversity of training and testing methodologies, and a lack of standardised support in the software domain. To improve compatibility with the ATHEENA front-end, minor alterations were made to the Early-Exit network source code so that the software versions of the Early-Exit networks could be used with the Early-Exit profiler and converted into a compatible form of ONNX. The original implementations of the networks targeted CPUs and GPUs for inference. This means that PyTorch (or other high-level libraries) would handle the scheduling of the dynamic network graph for execution. We narrow our focus to the PyTorch library but many similar functions exist in other Deep Learning (DL) libraries. ONNX generation from these libraries is typically accomplished with a single command and an example input to propagate the tensor shapes through each layer. The example tensor is used to generate the network execution graph, this means that the conditional operation based on the early exit decision will be evaluated, locking in the shape of the network. The resulting ONNX graph is automatically optimised using constant propagation to eliminate the conditional statement. To prevent the ONNX optimisation, we first translate the high-level model into an intermediate representation (IR) [87]. We can then leverage built-in methods to convert this IR to the final ONNX form, with all conditional operations explicitly included. Finally, we perform an inference using the high-level library and check that the generated ONNX representation produces the same result, thereby verifying that the conditional functionality has been preserved.

4.2.2 Early-Exit Profiler

To gather the required statistics of the sample difficulty distribution used to inform the design-time probability we develop a software profiling module called the **Early-Exit Profiler**, shown in Figure 4.1. This module takes as input a *profiling data set* and the high-level *Early-Exit ConvNet implementation*. The module attaches profiling trackers to the software-based ConvNet implementation and a series of batched inferences are carried out using the data provided. The trackers report the inference statistics for the network, specifically, the exit usage percentage, exit accuracy, and effective full network accuracy (taking into account where each sample would exit at run-time). The probability of the network encountering hard samples is profiled using a random subset of the test data [88] to then be provided to the optimiser as the value p . The ATHEENA optimiser then creates an optimised mapping for the different stages of the network according to the multi-stage CDFG hardware model, scaling the resource allocation subject to $1/p$.

4.3 Early-Exit Network Layers: Hardware Templates

We extended the ONNX parser of fpgaConvNet to support the operations required by the Early-Exit CNNs. These operations include:

- Softmax
- Reduction (`ReduceMax`)
- Numerical comparison (`GreaterThan`)
- Conditional (`If`)

Figure 4.2 illustrates the new Early-Exit layer templates responsible for the hardware implementation of these operations. Figure 3.1 provides a high-level view of the proposed placement



(4.2.1) As each input sample makes its way through the layer, the corresponding decision is passed to the Conditional Buffer along a control-flow path.

(4.2.2) We have indicated whether the input sample ID is kept (green) or dropped (red). The associated Sample data is either passed through or dropped.

Figure 4.2: New control-flow Early-Exit Hardware layer templates. The shapes adjacent to the arrows represent a given input sample ID.

of these layers within the pre-existing CNN data-flow operations in fpgaConvNet. The first three operations are the foundation of evaluating the exit decision. For simplicity these are merged into one hardware layer comprising of distinct modules. The conditional buffer layer does not directly correspond to the If ONNX operation, but supports the control-flow nature of the accelerator. The newly added layers match the fixed-point representation with the exception of the Exit Decision layer. This instead uses single-precision floating-point to preserve the numerical behaviour of the exponential function at its core.

4.3.1 Exit (Softmax) Decision Layer

Figure 4.2.1 provides a diagram of the behaviour of the Exit Decision layer. We represent each input sample's data as a different shape. An early exit will occur if Condition (4.1) holds for some threshold C_{thr} . This threshold is determined after training and will ideally result in a matching level of accuracy during network testing. In the diagram, we represent passing this threshold as the sample shape being coloured green. These decisions (coloured shapes) pass along a control-flow path to the conditional buffers which then drop or pass-through held data. The Exit Decision is responsible for evaluating against the threshold using the standard Softmax : $\mathbb{R}^C \rightarrow \mathbb{R}^C$ function which corresponds to (4.2). The Softmax function is an

exponential normalisation function that is provided with the raw output vector from the last layers of a DNN (or an early classifier). The softmax output (a.k.a. logit) is a vector in which all values are positive and sum to 1. As each of the raw values are exponentiated, this has the effect of significantly shifting the proportion towards positive values and diminishing negative values. This resultant vector is often treated as a probability distribution over the possible class values for the classification task [89].

$$\max_{i \in \{1 \dots C\}} [\text{Softmax}(x)]_i > C_{thr} \quad (4.1)$$

$$\text{Softmax}(x) = \frac{\exp(x)}{\sum_{j=1}^C \exp(x_j)} \quad (4.2)$$

C is the number of classes, x_i is the network output for the i th class, and C_{thr} is the confidence threshold.

The operation can be rearranged to remove the division as in (4.3) to decrease the resource overhead and latency of the decision component. The use of floating-point for this layer means that addition and comparison operations incur a significant latency penalty given a target frequency. To reduce this penalty, we implement adder and comparison trees to compute results in parallel, minimising the latency of this layer.

$$\max_{i \in \{1 \dots C\}} \exp(x_i) > C_{thr} \cdot \sum_{j=1}^C \exp(x_j) \quad (4.3)$$

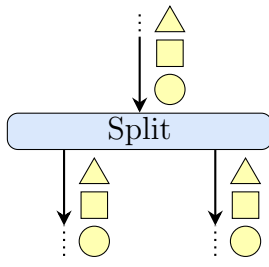
4.3.2 Conditional Buffer Layer

This component enables the resource scaling to be effective in improving the overall throughput of the accelerator. In Figure 4.2.2 we can see why. Input samples are being read into the buffer after undergoing some initial computation along the backbone of the network. The temporary

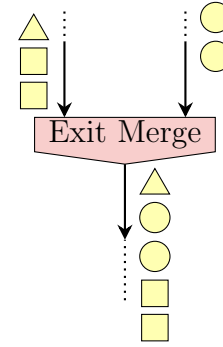
storing of these intermediate feature maps prevents unnecessary computation from occurring downstream at the later stages of the accelerator. Two challenges arise from the addition of control-flow from conditional buffers. The first challenge is that, after the first stage of the network, data samples will go down the path to the early exit or the path to the second stage. Let's assume N data samples of which, K will exit early and $N - K$ will continue through the second stage. The existing fpgaConvNet hardware layers in the second stage expect N data samples and the implementation of the pipeline will only terminate on the final N th data sample. To avoid deadlock, we flush the pipeline with an unused sample ID and corresponding data.

The second challenge is that there is a significant amount of computation between the point at which the network first has to buffer data and the point at which the control signals are produced. We temporarily buffer the unfinished data sample while the confidence metric is evaluated as well as the fully processed sample result at the early exit classification stage. This is accomplished in the example two-stage network with two conditional buffer layers.

If the sample can exit early, the buffer drops the intermediate feature map, while the classification result of the early exit is transferred to memory via the exit merge component (details in 4.3.4). Conversely, if the sample cannot exit, the buffer passes the intermediate data through to the next stage of the backbone. Due to the size of the buffer responsible for storing intermediate feature maps, it is important to reduce the impact of both reading in and writing out of the buffer (potentially doubling the latency of the layer for every sample, prior to the control signals). To drop an unused feature map we invalidate the addresses of the stored feature map in a single cycle. The fundamental operation of the conditional buffer is to filter the easy inputs from the hard inputs. The buffer prevents the later stages of the backbone from unnecessary computation, which has the effect of increasing throughput through the reduction in the required data rate of the second stage.



(4.3.1) Inputs to this layer are duplicated at the output.



(4.3.2) Inputs to this layer are merged into a single exit point before being transferred off-chip.

Figure 4.3: New data-flow Early-Exit Hardware layer templates. The shapes adjacent to the arrows represent a given input sample ID. The Exit Merge layer keeps all data for a given Sample ID sequential, opting to stall an exit instead of interleaving.

4.3.3 Split Layer

The split layer (Figure 4.3.1) is used to duplicate the result of layers at the branching points in the Early-Exit network. This splits the data-flow stream to allow a copy of the data to continue down the backbone in parallel to the early exit layers. This layer also unifies the back-pressure from both branches due to the valid/ready interface behaviour being applied from the branches upstream.

4.3.4 Exit Merge Layer

Inference is run on a batch of data samples where each data sample consists of a fixed number of pixels. In line with other static streaming architectures, the original fpgaConvNet has no built-in distinction between input samples in the pipelined hardware. Each component will continue to operate on newly provided pixels and the user is responsible for interpreting the results based on their location in memory. However, given that in an Early-Exit network, data samples within a batch are able to exit out of order, there needs to be an internal representation of each data sample's position within the batch. A *Sample ID* is assigned to each data sample and is passed through the hardware with each pixel. At the conditional buffers, the IDs are

compared to determine whether or not to drop the data points with a given ID. When a sample exits the network, the exit merge layer coherently merges the exit streams into one memory writing component. This may result in the stalling of one network path while another is allowed to pass through. We represent this in Figure 4.3.2 using identical shapes to represent pixels with the same sample ID. Square arrives at the component first so is read out to memory first, the first circle arrives to the component just after the last square so this sample starts to be read to memory. The triangle then reaches the merge at the same point as the next circle. Since the merge is coherent the triangle data is stalled while the circle data is read to memory before finally being read to memory as the last sample ID.

4.4 HLS Limitations & Parallel Compilation

One of the limitations of fpgaConvNet relying on hardware components templated in C++ for HLS, is that the scale of the networks that can be successfully compiled is heavily restricted. To combat this, we create helper functions in both Python and the Tcl scripting language that split the accelerator design into individual layers to perform separate, smaller HLS compilations that can be run independently and in parallel (subject to computing resources). These layers are then automatically stitched together at the board design stage within Vivado IP Integrator. Furthermore, we instantiate the supporting blocks for the accelerator. These include:

- DMA Controller for Off-Chip Memory Access
- Input & Output FIFOs
- Processing System & Peripheral Reset Controller

The multiple HLS IPs generated by the helper functions each require start signals from the host. Our updated host code generation automatically includes the correct commands tailored to the custom HLS IPs. In addition, the DMA read and write transfers are included with

the batch size set to the user’s specifications. By the end of the compilation, the user has a bitstream and complementary host code to perform batch inference on the board. We now detail the experimental results from using the toolflow to accelerate an Early-Exit network.

4.5 Experimental Results

We explore a case-study for the ATHEENA toolflow by implementing one of the original networks from the BranchyNet work [22]: B-LeNet. We chose BranchyNet because it was one of the first successful implementations of the Early-Exit strategy for reducing average computation while maintaining accuracy. The authors provide a detailed methodology for their implementation and training, supplemented with an open-source repository. This made it possible to accurately replicate the networks detailed in the paper and allow us to build on them for experimentation.

The hand-crafted nature of the networks and their Early-Exit placement is simple, so provides us with a good base-case example implementation of the style of networks that need to be covered by our toolflow. Compared to Early-Exit networks like MSDNet, the BranchyNet implementation does not require significant changes to be compatible with the fpgaConvNet components of our toolflow. Our adaptation started with some minor modifications to the source architecture, shown in Figure 4.4. The original network source-code was written in a predecessor to the PyTorch framework so we re-implemented a modern version in PyTorch to feed as input to the ATHEENA toolflow. The updated network was trained and tested in the manner outlined in the original paper. The alterations resulted in a negligible change in accuracy and a similar Early-Exit probability for a comparable confidence threshold (C_{thr}) value in the software domain. We also note, some of these modifications reduce wasted compute on the board and improve compatibility with standard fpgaConvNet convolution layers.

Following ONNX generation, the Early-Exit profiler then generates the ONNX representa-

tion and Early-Exit probabilities to inform the DSE process. This in turn creates a hardware mapping between the Early-Exit network and fpgaConvNet hardware layers with respect to chosen board’s resources. The tool then converts the mapping into HLS code and compiles the layers in parallel before stitching them together and generating a bitstream with associated host code.

4.5.1 BranchyNet: Hardware Experimental Study

We compare our hardware Early-Exit implementation to a corresponding single-stage network baseline optimised by open-source fpgaConvNet. This single-stage (backbone) consists of the network layers from the start of the Early-Exit network through to the end of the second stage. For BranchyNet, this means three Convolution, Pooling and ReLU layers followed by a Linear (Fully Connected) layer. The experimental setup follows that of the original fpgaConvNet paper in order to simplify comparison. The target device is the Xilinx ZC706 board [90] with the Zynq 7045 System on Chip (SoC). The resources available are 218600 LUTs, 437200 FFs, 900 DSPs, and 1090 18K BRAMs. Vivado HLS 2019.1 was used for compatibility with fpgaConvNet. Each design is conservatively clocked at 125MHz (limited primarily by HLS compilation).

Both the ATHEENA optimiser and baseline optimiser are provided the board resources constrained at different percentages in order to generate a Throughput-Area Pareto curve. Due to the random aspect of the simulated annealing within both optimisers, they are run ten times and the best points are chosen to form the curve. Additionally, a range of data points with constrained resources allows us to infer throughput gains/resource savings on boards with lower available resources. Once the optimisers generate the hardware mapping, we collate the predicted results shown in Figure 4.5.1 and then pass through the best performing subset of these results to the HLS backend in order to generate the board design. Figure 4.5.2 shows the resulting Throughput-Area Pareto curve after synthesis, implementation and place-and-route.

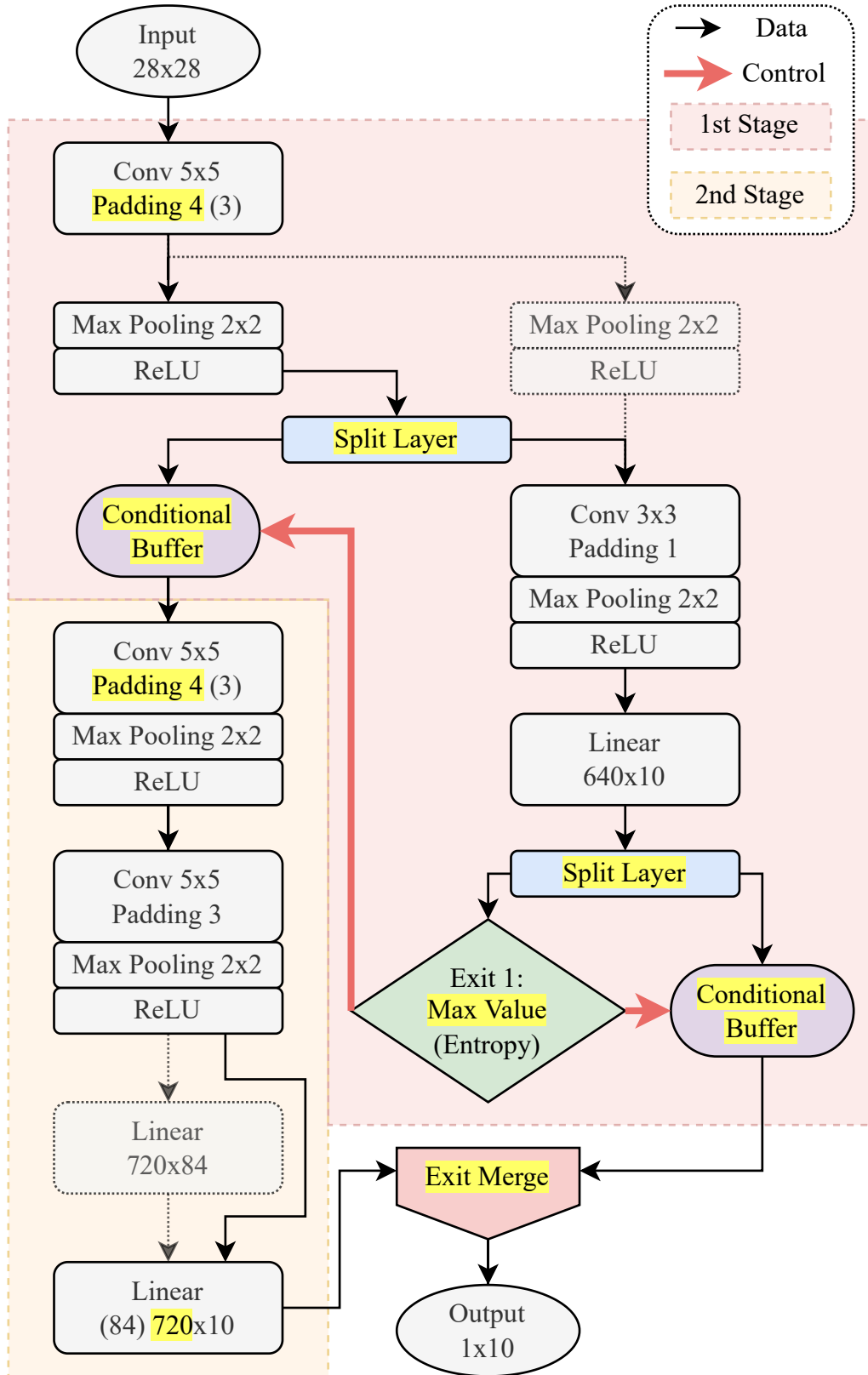
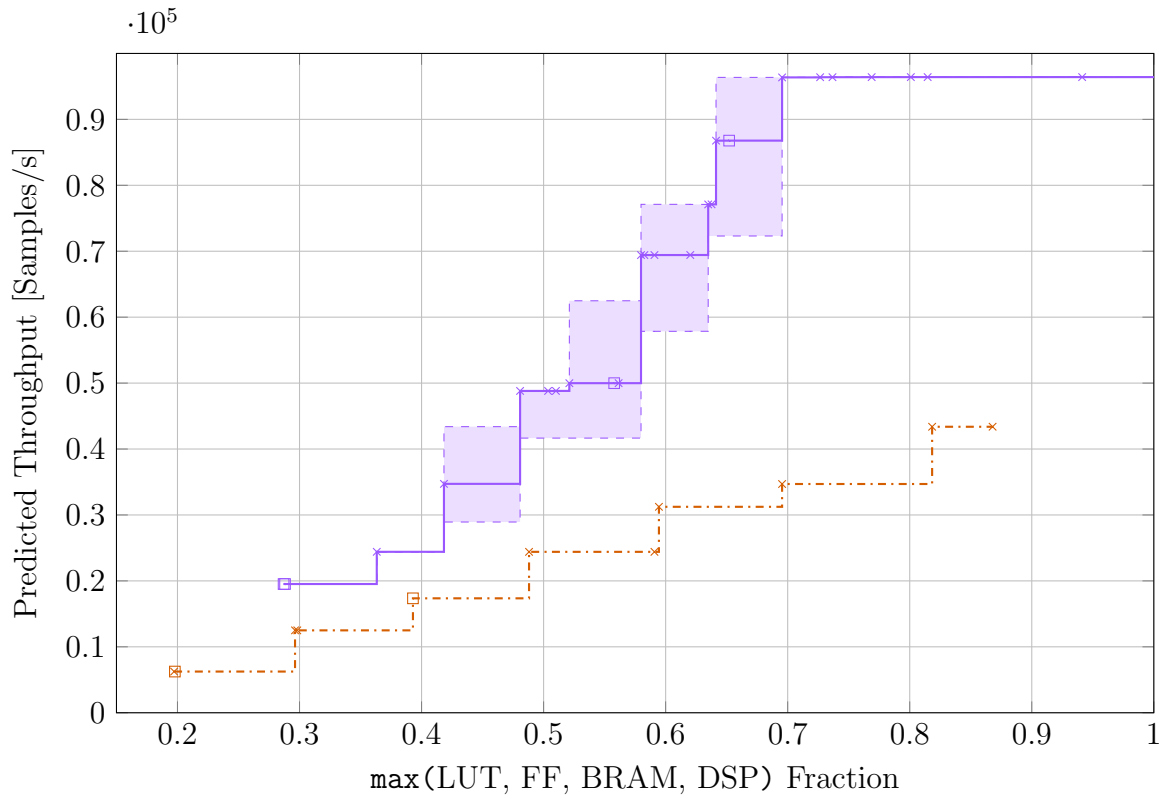
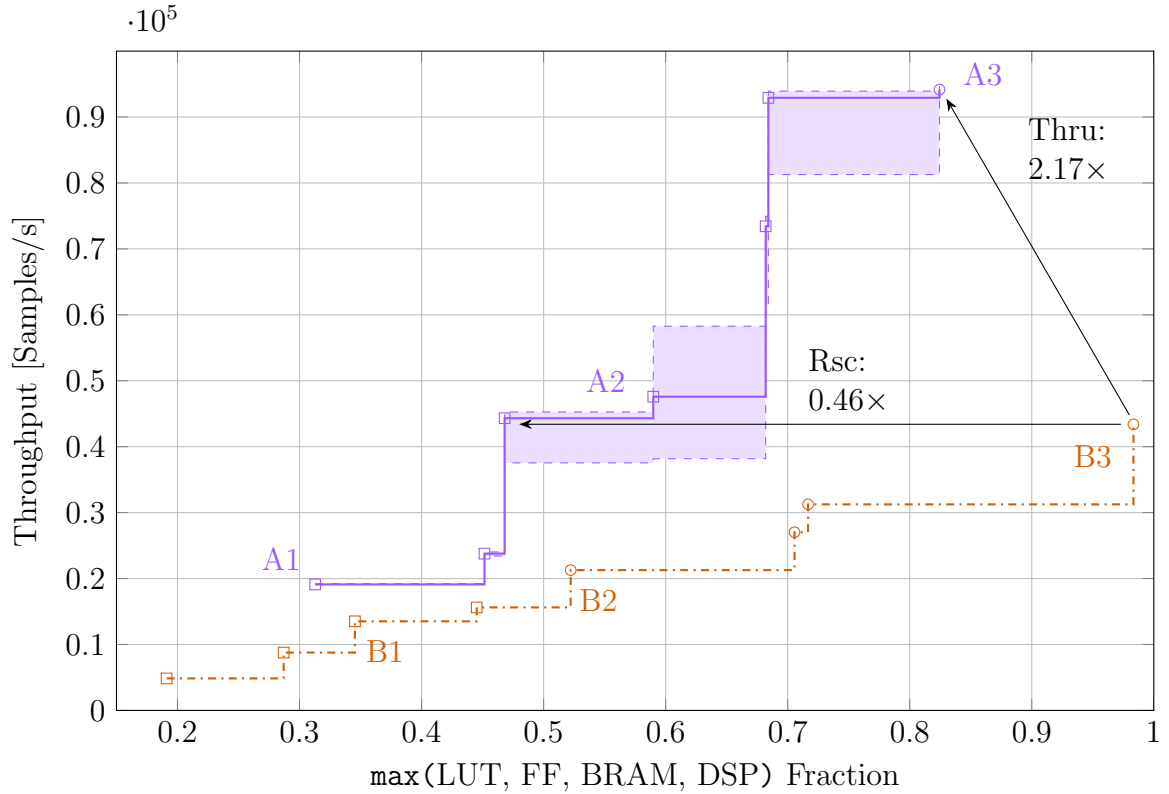


Figure 4.4: Slightly modified version of B-LeNet [22] for fpgaConvNet. Changes are highlighted in yellow with original values in brackets. Faded layers have been removed from the architecture. Hardware-only layers (detailed in Section 4.3) have been added.



(4.5.1) Throughput Area results generated by the simulated annealing optimizer stage of the toolflow at a p of 25% and $q = p \pm 5\%$.



(4.5.2) Throughput Area results obtained using ZC706 board. At a p of 25%. Tested using data sets consisting of $q = 30\%, 25\%, 20\%$.

Figure 4.5: The orange dashdot line represents the baseline results for *fpgaConvNet*. The limiting resources for each point are denoted by the following: $\times$ (BRAM), $\square$ (LUT), $\circ$ (DSP). Designs A1, A2, A3, and B1, B2, B3 have detailed resource usage in Table 4.1.

The resource usage is recorded and the board loaded with each bitstream. The automatically generated host code for the board loads a batch of 1024 samples onto off-chip memory and enables the DMA transfers to and from the design. We measure the total time taken from the start of the DMA transfer until the DMA block registers as being idle and use this to calculate the throughput of the network. The same DMA controller is present for baseline and Early-Exit implementations so the impact on resources is consistent for a fair comparison.

The baseline results are represented by the orange dashdotted lines in Figure 4.5. While the modelled results are not accurate on a point by point basis, the trend is consistent for the predicted and board implementations. To allow for a fair comparison, we included an implemented point predicted by the optimiser to consume greater than the boards resources, in practice the design point B3 consumed 98% of the DSPs. For higher resource allowances, we can see that the designs tend towards being limited by DSPs, this is due to the baseline optimiser selecting an increased level of parallelism for the Convolution and Linear layers and is common to both the baseline and ATHEENA board implementations.

The solid purple line shows the case when the q (percentage of easy samples in the test set) matches that of p (the design-time probability), in this case $q = p = 25\%$. The dashed lines represent a range of points taken on the Throughput-Area space with a differing percentage of hard samples. The predicted results are calculated under ideal conditions: assuming sufficient buffering (determined through experimentation) and a regular sequence of ‘easy’ and ‘hard’ samples that directly match the exit probabilities being tested. For example, a $q = 25\%$ would result in a test set of 100 samples containing 25 ‘hard’ samples. This results in the second network stage being able to achieve the estimated throughput.

For the implemented board results, we sample a batch from the test data set for the classification task. The sampled test set has a split of easy and hard samples proportioned according to the required test probabilities but distributed randomly within the batch of 1024 samples.

The lower dashed line represents $q = 30\%$ and demonstrates a partially reduced throughput for some of the data points. This behaviour is expected when the distribution of the test data varies from the profiling data used at design-time to inform the DSE. The upper dashed line represents $q = 20\%$ and shows that some mismatching profiling situations can still result in throughput increase thanks to the significant latency reduction of data samples exiting at the higher throughput of the first stage. The results demonstrate the potential for greatly improved throughput at the risk of slightly decreased throughput. Primarily resulting from differences between test and profiling exit probabilities, and variation in exit distribution within a batch. Additionally, the predicted Throughput-Area curve is a good approximation of the measured implementation results. We can see the ATHEENA model is slightly optimistic in terms of achievable throughput. This is likely due to sub-optimal resource models for the new components and the variability of HLS compilation. Finally, the gap between the lower dashed line and the baseline is indicative of the robustness of the approach to the difference between p and q for a real world application. We note that these results are possible under the assumption that the buffering between stages is sufficiently large, a challenge we address in Chapter 5.

The maximum measured ATHEENA throughput is $2.17\times$ the the maximum measured baseline throughput. ATHEENA achieves this throughput using 16% fewer DSPs (the limiting resource) when the resources are apportioned between the first and second stage according to the profiled probabilities. Alternatively, ATHEENA can achieve the same throughput as the maximum baseline using 46% of the design’s limiting resource.

The design points in Table 4.1 show that the achieved throughput increase comes at the cost of an increase in BRAM proportion. The designs A1, A2, and A3 (labelled in Figure 4.5.2) require this as part of the conditional buffers to store enough of intermediate feature map samples to minimise stalling.

Table 4.2 details the resource overhead for the labelled designs in Figure 4.5.2. The per-

Table 4.1: Tabulated resource comparison for implemented Baseline Vs ATHEENA on ZC706 board. *Maximum limiting resource.

	LUT	FF	DSP	BRAM	Limiting Resource (%)	Throughput (Samples/s)
B1	75513*	61361	295	55	35	13513
A1	68383*	63170	169	206	31	19434
B2	105451	84761	470*	89	52	21276
A2	128940*	117138	407	239	59	47583
B3	138194	120764	885*	170	98	43384
A3	163637	142913	742*	265	82	94170

Table 4.2: Resource overhead of the Early-Exit for labelled designs compared to the network backbone.

	LUT	%	FF	%	DSP	%	BRAM	%
A1	13912	20	13595	22	34	20	114	55
A2	37766	29	35941	31	122	30	146	61
A3	33166	20	30974	22	112	15	186	70

centages correspond to the fraction of the total design resources attributed to the Early-Exit hardware layers. This includes the resource usage attributed to the *additional* Early-Exit computation and buffering required for operation of B-LeNet. The proportion of this overhead is detailed as a percentage of the total design. The overhead contribution is from implementing the additional classifier layers, comparison layer, and conditional buffering layers, all occurring in the first stage of the accelerator. In each case, the BRAM usage is the most significant proportion of the Early-Exit resource overhead, with design A3 having 70% of the total BRAM usage within the Early-Exit overhead. This highlights the importance of the conditional buffer in the efficacy of our method but also indicates that the buffer sizing was larger than necessary for some design points. We note that designs A1 and A3 are demonstrably less variable in terms of throughput, this is unlikely to be solely due to the large buffer sizes since A2 is also dominated by BRAM. The predicted throughput of each stage of the Early-Exit network is calculated separately. The resulting design points for each stage form a discrete Pareto front. When the optimiser is selecting from these two stages and scaling the throughput of the second stage, there will often be a discrepancy between the predicted throughputs of the stages. If the

Table 4.3: Comparison against BranchyNet reported accuracy, converted throughput and converted hard sample probability.

Network		Top 1 Acc. (%)	p (%)	Throughput (Sample/s)
CPU	LeNet	99.20	-	297
CPU	B-LeNet	99.25	5.7	1613
GPU	LeNet	99.20	-	633
GPU	B-LeNet	99.25	5.7	2941
Baseline	LeNet	98.84	-	43384
ATHEENA	B-LeNet	98.88	25.0	94170

resulting combined design point over-provisions the second stage then the design will be more robust to variations in the testing data set probability. Excluding the BRAM usage, we can see that the Early-Exit resource overhead is higher for design point A2, suggesting that the throughput is more tightly coupled to the performance of the first stage of the network.

The original BranchyNet paper details an implementation of the LeNet and Branchy (B) LeNet networks using using a 3.0GHz CPU with 20MB L3 Cache and NVIDIA GeForce GTX TITAN X (Maxwell) 12GB GPU. They report the average latency of a single sample in milliseconds. We convert their per sample average latency metric into a throughput metric for the comparison. Both our baseline and ATHEENA designs benefit from adaptations of the network architecture to be more amenable to a hardware implementation. These changes include quantisation to a fixed-point representation, adjusting layer parameters, and adapting the exit condition computation. This has a marginal effect on the accuracy compared to the software implementations and is partly the reason for such high gains compared the CPU and GPU implementations.

We are able to exploit per-sample parallelism more effectively using the streaming architecture however the relative gains of implementing Early-Exit on CPUs and GPUs is greater than demonstrated by our toolflow in part due to necessary reduction of exit percentage to maintain accuracy. This drop in exit percentage is due to the limitations of the fixed-point representation of the threshold value coupled with the high degree of classification accuracy on

this particular dataset. This means that the possible values for the threshold on the FPGA hardware are limited. We found that the threshold required to get the same p value and maintain the accuracy could not be implemented on the FPGA at the lower precision. We chose to reduce the exit percentage to maintain accuracy at the cost of potential gains in throughput. Despite this, we find that a value as low as $p = 50\%$ still results in a throughput improvement relative to the baseline in spite of the area overhead for the additional layers and control logic embedded in the Early-Exit streaming architecture. Overall, significant gains in throughput can be achieved from utilising an optimised streaming architecture, implementing a CNN on an FPGA, and up to $2.17\times$ further gains from implementing customised Early-Exit hardware tailored to the design.

4.5.2 Benchmarking Results

To support the case-study, we provide some additional benchmarks in Table 4.4. The best performing predicted throughput and corresponding resource results have been taken from the optimiser stage of the baseline and ATHEENA toolflows. Due to the increased size of these networks we specify the target platform as the Xilinx VU440. We use the percentage of hard samples outlined in the papers to generate the two-stage design.

Table 4.4: Resulting throughput improvement for two-stage accelerator designs generated by ATHEENA model compared to *fpgaConvNet* baseline. *Implemented on Xilinx ZC706

Network (Task)	Toolflow	Limiting Resource Type	%	p (%)	Throughput (Samples/s)	Gain
B-LeNet[22] (MNIST[88])	Baseline*	DSP	84	-	43384	$1.00\times$
	ATHEENA*	DSP	88	25	94170	$2.17\times$
Triple Wins[91] (MNIST[88])	Baseline	DSP	86	-	19524	$1.00\times$
	ATHEENA	DSP	81	25	54220	$2.78\times$
B-AlexNet[22] (CIFAR10[92])	Baseline	DSP	84	-	8676	$1.00\times$
	ATHEENA	DSP	88	34	17357	$2.00\times$

While we focus on the smaller networks from the BranchyNet paper, they also provide a large ResNet implementation operating on the ImageNet dataset. Using their exit probabilities, our

approach should generalise since they detail a significant proportion of inputs that exit early, highlighting that a performance increase *is* possible provided the use of a large enough FPGA.

To further support the generalisation, and as part of ongoing research for the TCAD submission (Chapters 5 & 6), we have looked at implementing small ResNet networks used in the MLPerf Benchmarks [93] for the CIFAR10 [92], CIFAR100 datasets, and Tiny ImageNet [94]. These are much larger than the MNIST dataset in our case-study and, while not achieving as great a performance gain, these networks were able to improve accuracy over the static network implementation and increase throughput, according to our toolflow model.

4.6 Summary

In this chapter, we have demonstrated the benefits of the *input-dependent* computation paradigm in improving CNN mapping to FPGAs by implementing a toolflow that allows for the exploration of the throughput/resource trade-off space of Early-Exit network hardware implementations. The successful implementation of the methodology in Chapter 3 required extending an existing toolflow with control-flow representation and the development of specific Early-Exit layers that can handle the intermediate buffering and conditional data-flow. Finally, we verified the toolflow’s model of predicted performance by implementing multiple, resource-constrained, design points on an FPGA board with randomised test samples. The robustness of the approach is explored using adapted test sets with known Early-Exit probability variation. In the next chapter, we will explore a key assumption made about buffering and the journey towards a more accurate design model for the accelerator’s probabilistic nature.

CHAPTER

5

Queueing Theory for Throughput Estimation with Finite Buffers

This chapter addresses the assumption tying performance to ‘sufficient buffering’ by restating the resource scaling methodology and deriving a more accurate model of throughput, incorporating buffer sizing. This work has been submitted to the IEEE journal: Transactions on Computer Aided Design.

5.1 Introduction

Having an accurate estimate of throughput is vital to determining the optimal design points with respect to constrained resources. The resource scaling methodology from Chapter 3 relied on the assumption that the amount of buffering between the network stages would be large enough to mitigate deviation from the profiled probability of ‘easy’ and ‘hard’ inputs. This allows the tool to scale the throughput of the second network stage by the probability that it will be used, i.e., the probability of a ‘hard’ input. The resulting throughput of the entire accelerator is then limited by the throughput of the first stage, or the scaled throughput of the second stage. This two-stage approach to determining the throughput decouples the behaviour

of two network stages when in actuality the inputs passing through the second stage directly affect the throughput of the first stage through pipeline stalls. The probability that a stall in the second stage provides back-pressure upstream to stall the first stage is a function of the probability of encountering a ‘hard’ input, the time it takes for the second stage to be able to process the next input (initiation interval), and the size of the buffer or queue linking the two stages. The probability that the finite queue and the second stage stall the first is the blocking probability p_B .

The difficulty with a large buffer size is that entire feature maps need to be stored on-chip, meaning that the majority of BRAM will be consumed, limiting the memory available for the DNN parameters, reducing the throughput of the accelerator. However, too small a buffer between stages would mean that consecutive ‘hard’ inputs would result in the second stage stalling the first and drastically reducing the overall accelerator throughput. While impossible to completely eliminate the chance of stalling, it can be significantly reduced by having some level of buffering. The challenge then becomes accurately estimating the effect that buffer size has on the throughput of the accelerator, given the finite nature of the resources on the FPGA – specifically BRAM. This chapter introduces an abstraction of the accelerator hardware that can be modelled using Queueing Theory. The first level of abstraction results in the creation of a simulator which can predict throughput under different buffer sizes but at a high computational cost. Increasing the level of abstraction resulted in multiple interpretations of the accelerator behaviour as a queueing system to varying degrees of accuracy. Queueing Theory provides key assumptions for these systems as well as avenues for estimating the throughput of each of these systems. These approaches allow us to reduce the computational complexity while maintaining accuracy, resulting in a method of throughput prediction that is provably accurate and can guide the DSE engine to the optimal designs. The following section details how we abstract the complex components of a full accelerator system into a simpler model that allows us to make

reasonable assumptions on the stochastic nature of the data flowing through the system.

5.2 Accelerator Abstraction

We assume the accelerator on the FPGA device is part of a full system. Since we are targeting high throughput, we can reasonably assume that this system is designed to receive classification or task requests in aggregate from multiple user interfaces. The requests from different users are independent from one another so can be characterised by a single value determining the average rate at which the classification requests are fed into the system. The input (or arrival process) is defined by this rate (ϕ) and can be effectively modelled as a Poisson process, as this is a realistic way to represent the aggregation of requests from a multi-user system. Receiving the inputs, is the unbounded queue which represents the off-chip memory. From which the inputs are then transferred to the first stage of the accelerator.

Representing the accelerator at the layer or module level introduces a significant amount of complexity due to the pattern through which data streams in and out of each component. Therefore, as with the preceding chapters, we group the layers into different exit stages.

In Queueing Theory terminology, these stages become the first and second processing node. Having condensed the all layers into a handful of nodes, it becomes unnecessary to model the throughput of an individual pixel. As such, the inputs or tasks in the queueing system are full inputs. The dynamic nature of a two-exit ATHEENA accelerator can then be abstracted as the queueing system shown in Figure 5.1.

This first stage is modelled as a process operating at the higher throughput with a deterministic service (execution) time. The service time of the first stage corresponds to the initiation interval of slowest (lowest throughput) layer *within* the first stage. The conditional buffer component sits between the first and second stage. The conditional buffer operates in the same fashion as outlined in Section 4.3.2, storing intermediate feature maps temporarily as the exit

decision is calculated in parallel. In terms of the abstraction, the storage behaviour of a FIFO constructed of BRAM corresponds to a finite queue. The size of the queue (k) is calculated by taking the amount of BRAM available to the buffer and dividing it by the number of bits required to store the intermediate feature maps (partially computed results).

For example, if a single feature map consisting of a 100×100 array of 16 bit words can be stored in 10 18k BRAM blocks, then an available buffer size of 10 BRAMs would have a queue size of $k = 1$. A buffer size of 20 BRAMs would have a $k = 2$ and so on.

Once an input sample reaches the head of the finite queue, it will either be dropped (in the case of an early exit) or be processed by the second stage, taking a fixed amount of time corresponding to the initiation interval of the second stage. The likelihood of these events is dictated by the design-time probability of encountering an easy/hard input, allowing us to construct a service time distribution for the second process. Assuming the system has settled into a steady-state, the second stage can be modelled as a *two-point distribution*. This distribution is a generalised form of the binomial distribution, having probabilities for its random variable to take on one of two values. This distribution can be characterised by mean and variance values. For a network of more than two exits, the form of the distribution can be extended to a multi-point discrete distribution as an approximation for multiple exits.

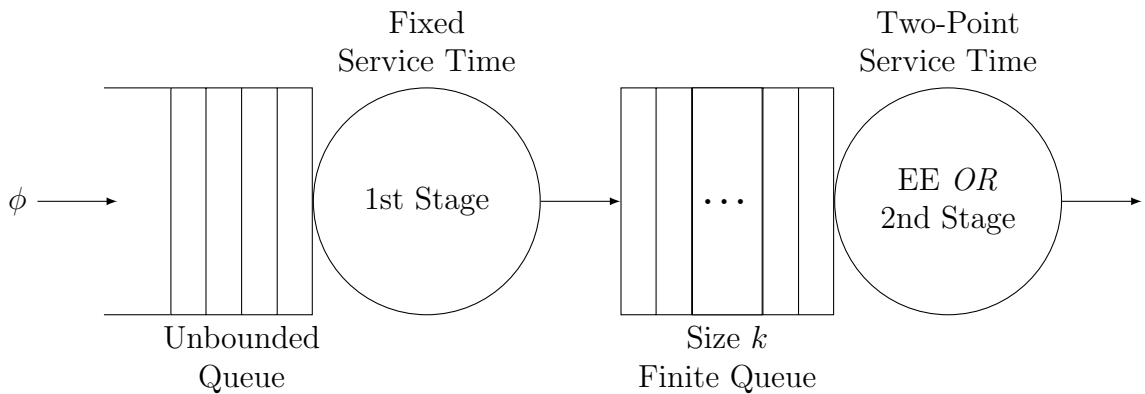


Figure 5.1: Queueing system abstraction of the EE-DNN accelerator, containing two processors with differing service time distributions, linked by a finite queue.

We have created an event-driven software simulator to model the abstracted queueing system in Figure 5.1, and to determine the throughput of different designs under finite buffer sizing.

Each simulation run must first reach steady state operation before throughput metrics can be extracted. A single simulation instance must process 100,000 samples and be repeated 50 times to achieve the necessary standard deviation between 0.2–0.6% of the throughput fraction. This means that exploring the design space for a *single* queue value, a simulation can take on the order of 200 seconds, despite the use of an optimised, multi-threaded software implementation.

To highlight the issue with simulation, the original DSE engine was profiled and a run takes approximately ≈ 0.5 s. From this run, 23% is spent on determining the best combination of the network stages, in which the original, naive throughput calculation accounts for 1.6%. Replacing the naive throughput calculation with the simulation that *accurately* takes into account the effect of the finite buffer size causes the DSE run-time explodes from ≈ 0.5 s to 5×10^6 s (≈ 55 days). As such, the simulation approach is impractical for the DSE used in the ATHEENA toolflow. Therefore, to perform the DSE in a reasonable time-frame, the simulation accuracy needs to be reproduced with a computationally efficient analytical method.

5.2.1 Analytical Models

We propose the use of the simplified Markov model in Figure 5.2, where p_B denotes the blocking probability and K the size of the finite queue, i.e., buffer. By collapsing the two processor model into a single processor, we can further approximate the buffer behaviour of the simulated model by abstracting the two-stage queueing network as a Markov chain with $K + 1$ states, mapped to the blocking ($K + 1$) and non-blocking ($0 \dots K$) states. The standard approach when modelling queueing systems is to assume that both arrival *and* service times follow a Poisson distribution [95]. This model is described as $M/M/1/K$, where M denotes the Poisson distribution, 1 the number of processors, and K the finite queue size. In a perfectly balanced system, both the arrival and service rate are equal – this would be equivalent to a first and second design stage with matching throughput values. In reality, the discrete nature of the

throughput values of the designs generated by the optimizer mean that these rates are unlikely to match, resulting in variability. We can incorporate this at design-time to improve the throughput estimation.

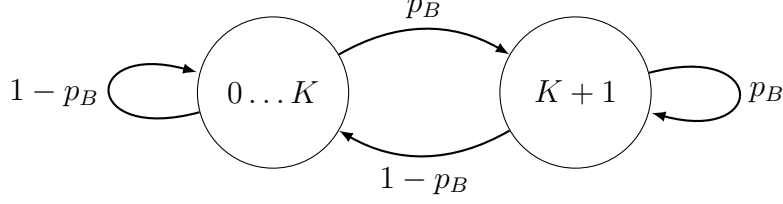


Figure 5.2: *Simplified Markov chain representation of a finite queue with capacity to store K items. p_B is probability with which the system enters the blocked state.*

The issue with the simple Markov method is that, as previously discussed, the second stage processor follows a two-point distribution. A key difference between the Poisson and the two-point distributions is their variance. By definition, the squared coefficient of variation (s^2) of a Poisson distribution is 1, whereas, the s^2 of the two-point distributions generated from the tested design-time probabilities can be as high as 3 or 4.

$$s^2 = \frac{v}{\mu^2} \quad (5.1)$$

The s^2 value can be calculated as above, using the mean (μ) and variance (v) of the service time distribution. If we want to further improve the accuracy we can base our second process distribution on the two-point distribution. However, using both the two-point distribution and the finite queue means the simple Markov chain model is no longer viable as the system no longer satisfies the memory-less property. This increases the complexity of analytically determining the blocking probability p_B . Therefore, we use an approximate method labelled $M/G/1/K$. This relies on the two moment approximation to derive the blocking probability from a generalized distribution [96] of service times and is based on the characteristics of the two-point distribution. The approach, outlined in (5.2) [59], approximates an embedded Markov chain representation using s^2 as a measure of variance to parameterise the general distribution.

$$\begin{aligned}
y &= \sqrt{\rho}(s^2 - 1) \\
p_B &= \frac{\rho^{\frac{y+2k}{y+2}} \cdot (\rho - 1)}{\rho^{\frac{2(y+k+1)}{y+2}} - 1}
\end{aligned} \tag{5.2}$$

ρ is the traffic intensity of the system so measures the ratio of the rate at which feature maps enter the finite queue and the rate at which feature maps are processed. To provide a single value for the second stage, the mean of the service time distribution is used. k is the size of the finite buffer. The main disadvantage of this method is that the accuracy of the approximation is reduced when the variance of the distribution increases.

An alternative to the embedded Markov approximation is the Quasi-birth-death (QBD) process. This can model a queueing system with two sequential processes, linked by a finite queue in which the second process can block the first. We can calculate the blocking probability from this model by creating a nested state transition matrix that contains the probability of increasing/decreasing the values in the queue and whether or not the second process is blocking. The Matrix Geometric method [53] can be used to determine the point at which a queueing system becomes unstable. We can then derive the maximal input throughput for the system while maintaining stability of the state transitions. The benefit of QBD is that it does not require the collapsing of the first and second stage into a single process node so is theoretically the closest to the original abstraction. However, this method suffers from the same Poisson distribution assumption for the arrival, and both processor nodes' service times as the $M/M/1/K$ methods.

The methods outlined thus far each have compelling theoretical reasons for their use in our toolflow as well as assumptions that will limit accuracy. In the following section, we perform a quantitative analysis of our chosen $M/G/1/K$ approximation, in addition to the other Queueing Theory methods, to demonstrate the performance relative to simulation.

5.3 Verification: Accuracy & Computational Analysis

We compare the $M/G/1/K$ approximation with several other standard Queueing Theory methods, with varying accuracy and computational requirements in order to verify the suitability of our chosen method. Table 5.1 provides a quantitative view of the available methods. The timing results for each method were profiled using the cProfile and timeit Python libraries [97], [98]. Each method was run 100 times in succession and this repeated 5 times for 18 queue values and 5 differing probabilities. We report the average and worst case times of each of the methods in seconds for a single call to the function for one queue value. We time the methods as a part of the ATHEENA component of the DSE, summarized in the DSE Average column. This is a measure of the total time for combining the network stages, including the calls to the methods. The cumulative error reported is the sum of the absolute error of each queue value, for each tested probability, compared to the throughput fraction reported in simulation. The general trend for error is heavily dependent on queue size so an average value would be heavily skewed. We used the cumulative error value to avoid implying consistent error across queue values. The worst case error is the maximum absolute error for an individual queue size over all the tested probabilities. We use this value to give an indication of the maximum deviation from the simulated probability.

The $M/M/1/K$ balanced and variable, and the $M/G/1/K$ approximation methods are scalar-based formulas so have relatively similar compute times, significantly faster than that of the simulation. Since the balanced method assumes an ideal ratio of arrival and service rates, the error drastically increases for the 4 probabilities that deviate from optimal. The variable method compensates for this by factoring in the ratio.

The $M/G/1/K$ iterative method generates the full state transition matrix for the embedded Markov chain, incorporating the general two-point distribution's high variance and embedding the resulting transition probabilities to be converged using iterative multiplication. The two

Table 5.1: Quantitative view of the different Queueing Theory-based models. *(Estimate, due to excessive runtime)

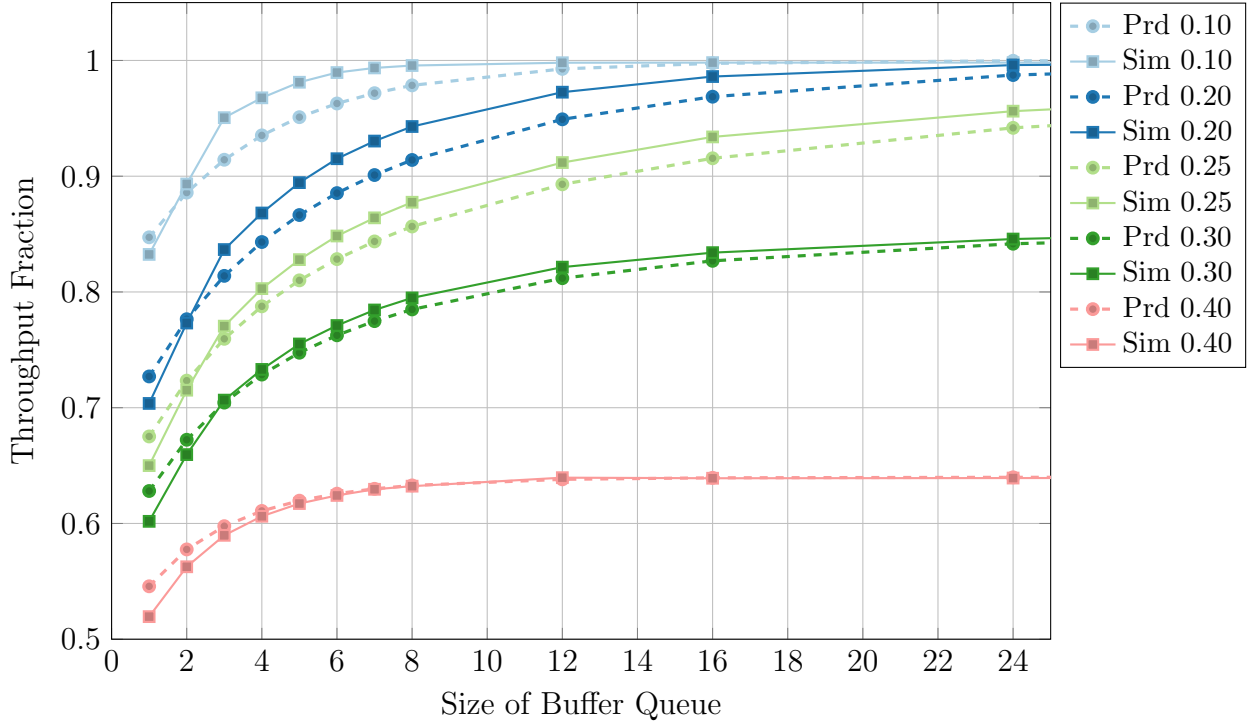
Method	Compute Time (s)			Error	
	Profiled Function Average	Worst Case	ATHEENA DSE Average	Cumulative	Worst Case
MM1K:					
<i>Balanced</i>	3.15E-07	1.37E-09	0.79	9.47	0.36
<i>Variable</i>	7.11E-07	7.03E-09	0.82	1.40	0.14
MG1K:					
<i>Approximation</i>	2.46E-06	2.70E-08	0.91	0.86	0.04
<i>Iterative</i>	1.54E-02	2.65E-03	1.10	2.93	0.11
QBD:					
<i>Matrix Geometric</i>	1.03E-03	5.21E-05	19.53	1.30	0.07
Simulation:					
<i>Event-Driven & Multi-threaded</i>	1.78E+02	1.86E+02	4.81E+06 *	-	-

main disadvantages of this method are that it breaks down for hard sample probabilities that result in an arrival and service time ratio above 1, and that the computation time is dependent on the queue value, which dictates the size of the transition matrix.

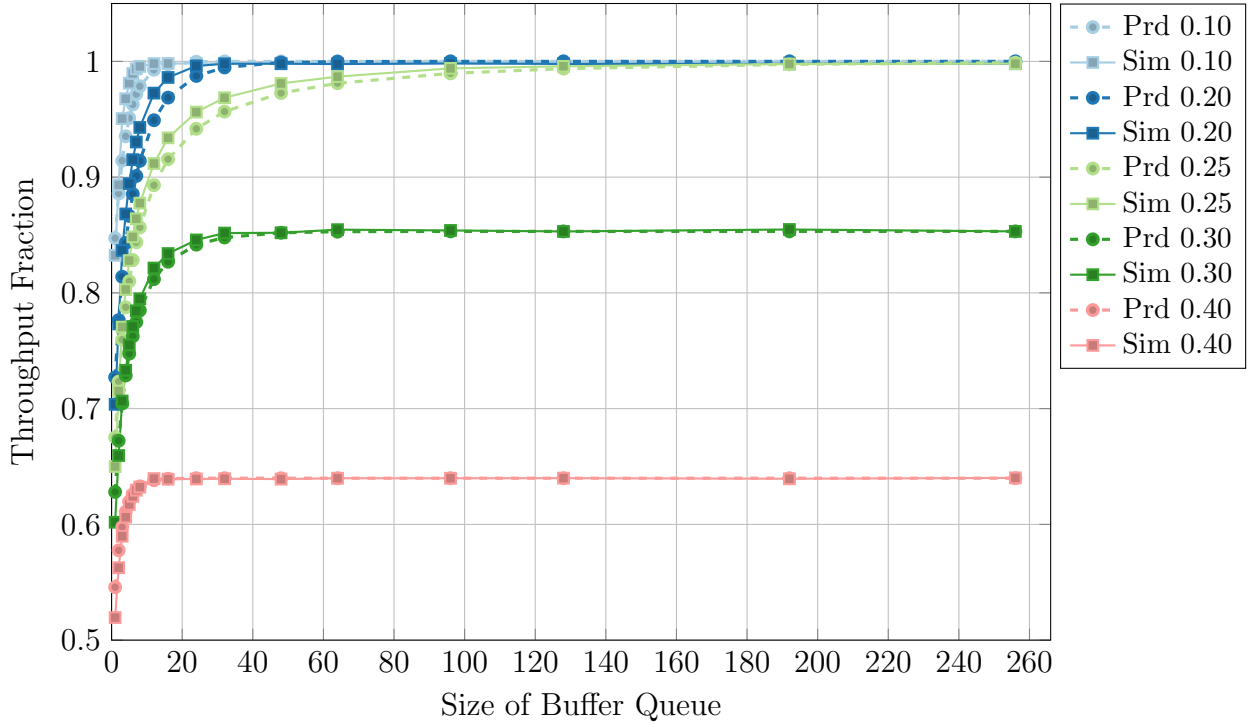
The QBD method also depends on a state transition matrix for its solution, leading to higher compute times. We can see that there is an accuracy improvement relative to the $M/M/1/K$ methods, due to the inclusion of multiple processors and blocking behaviour. However, the reliance on Poisson distributions for these processors prevents this from being as accurate as the $M/G/1/K$ approximation.

To verify the function run-times, we profiled the ATHEENA DSE using the chosen analytical method on the same set of network stages and found that approximately 0.060s of the total processing time of 0.91s was spent on the analytical method, which lines up with profiled average compute time. All the analytical methods significantly reduced the run-time of the ATHEENA DSE when compared to the simulation. The $M/G/1/K$ approximation reduces the time to below one second. Furthermore, Figure 5.3.2 demonstrates that the approximation method follows the asymptotic trends of the simulated values at higher queue values and Figure 5.3.1 shows that, in general, the method underestimates the final throughput fraction

at lower p values with a significantly higher coefficient of variation. This is where we observe the limitations in the *approximation* of the general distribution as part of a blocking queueing system. This reduction in accuracy in situations with a high coefficient of variation is noted in the original work [59]. The fact that these errors are an underestimation means that the model will, in the worst case, be over-provisioned in terms of resources. In spite of this small decrease in accuracy, the $M/G/1/K$ approximation proved to be the best choice for the ATHEENA toolflow as this method results in the lowest error at a manageable computational cost.



(5.3.1) Plotting predictions based on queue size of N . Focused on lower queue values to show trend in BRAM limited environments.



(5.3.2) Plotting predictions based on queue size of N . Expanded to show trend across all queue size values tested.

Figure 5.3: Plotting simulated and predicted curves for fractions of throughput at different second stage utilisation probabilities (p values). This is the fraction of the input rate throughput, the inverse of the limiting layer's initiation interval in the first stage of the accelerator. 'Prd' and 'Sim' correspond to the predicted values using the $M/G/1/K$ approximation method and the values reported by simulation at the specified p .

5.4 Resource Allocation Under Finite Buffer Constraints

The central challenge addressed in the updated methodology is how to automatically produce performant hardware that leverages the benefits of custom FPGA streaming architectures and the input-dependent, dynamic nature of the Early-Exit networks under the more stringent resource and latency constraints inherent to the task and platform setting. Our toolflow allows us to explore the standard, flexible FPGA design space augmented by probabilistic information about a given network and task. The goal of the toolflow is to allocate the finite computational and memory resources of a given FPGA to maximise throughput for an expected distribution of input samples with variable difficulty, accurately modelling the impact of the probabilistic nature of the accelerator under the Early-Exit task in order to inform the design choice. Having outlined the chosen approximate method for modelling the blocking behaviour, the remainder of this chapter looks at how to incorporate this into the toolflow DSE and explores the behaviour of the approximation in relation to the original method.

As in Section 3.2, the definition of our TAP function as one that is (non-strictly) monotonically increasing in each of its arguments. This function represents the maximum theoretically achievable throughput for each network stage for a given fraction of the total available board resources. Once again, these functions are generated with carefully constructed calls to the `fpgaConvNet` optimiser, specifying limited fractions of the board resource constraints. The resulting design points for each resource fraction and network stage are then collected for input to the ATHEENA optimiser. One key difference with the previous methodology is that, instead of a naive scaling of the throughput with respect to the design-time probability p , the mean of the second stage throughput is used to provide the new second stage TAP function. We illustrate this graphically in Figure 5.4. The two TAPs are then merged into a single TAP for a given Early-Exit network. This has been illustrated in Figure 5.5. The second key difference from the previous methodology depicted in Figure 3.2, is that the combined TAP function is

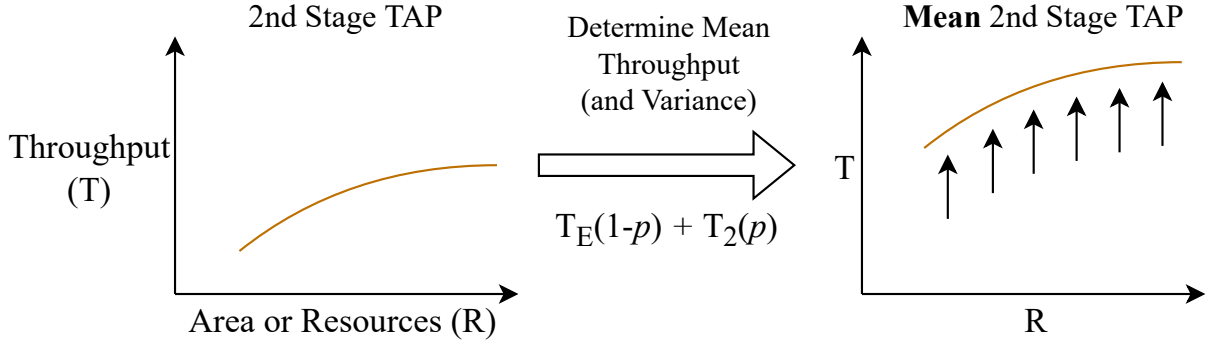


Figure 5.4: *Determining the Mean and Variance of the throughput of the second stage. For any given point on the TAP curve of the second stage, the mean throughput is determined by averaging this value, weighted with the probability of utilising the second stage (p), and the throughput of an early exit weighted with $(1-p)$. This has the effect of increasing the throughput of the second stage in a similar fashion to the scaling in Chapter 3.*

adjusted with respect to the size of the finite queue k .

The framing of ATHEENA's task is as follows: the second stage is only expected to be used by each input sample with some probability $p \in (0, 1]$, this means that it is possible to obtain a higher throughput than the nominal throughput of this stage, by a factor of $1/p$. However, since the sufficient buffer assumption has been removed, the second stage can only be scaled by a maximum factor $1/p$. To estimate the throughput for this second stage, we compute the weighted average between the throughput of an early exit and the nominal throughput of the second stage.

The throughput of a given combination of design stages is then approximated by the throughput of the first stage, scaled with respect to the likelihood that the second stage stalls the first, i.e., the blocking probability p_B determined through the two moment approximation [59]. In any practical setting, the stochastic nature of the distribution of easy and hard samples will result in deviation from the profiled probability for which the hardware was designed, resulting in some under-utilisation of the second stage. The profiled, design-time probability estimate is defined as p and the run-time (real-world) probability is q (with corresponding design and run-time *blocking* probabilities p_B and q_B). Amalgamating these definitions and (5.2), we are able to construct an ideal combination of the two TAPs, parameterised by p , q , k , and expressed

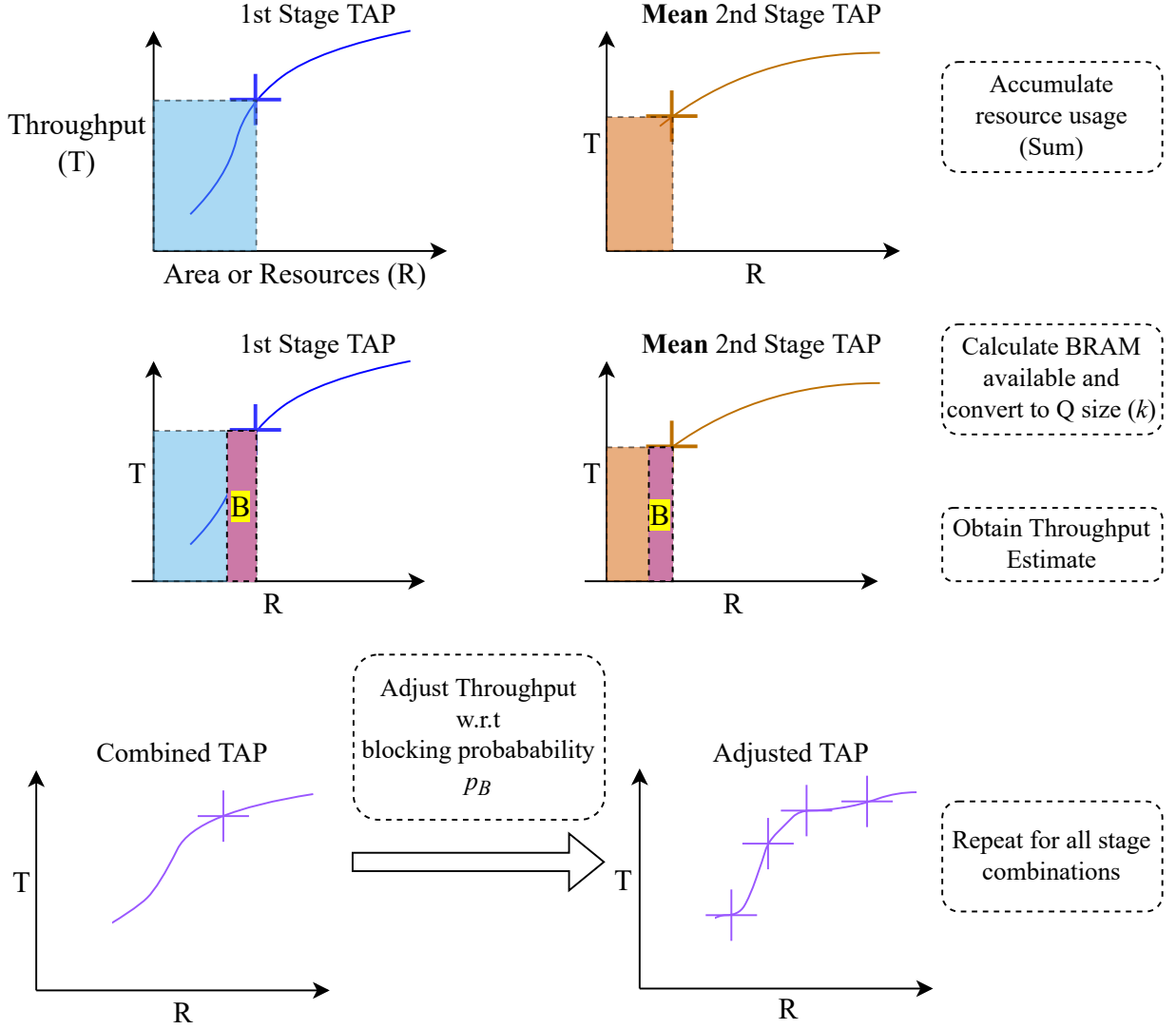


Figure 5.5: The functions depicted in the top two plots are Throughput-Area Pareto functions that represent the throughput of each network stage under a given resource constraint. To combine two stages for a full accelerator, first accumulate their resource values. If the maximum limiting resource is not BRAM¹, then allocate the remaining proportion of the BRAM so as to balance the resource fraction against the maximum resource. This BRAM is allocated to the finite buffer and has a corresponding queue size k (dependent on the intermediate feature map size as detailed in Section 5.2). This k value, along with the second stage probability p , and the first and second stage throughputs, are directly fed into the $M/G/1/K$ approximation to determine the throughput of the combined design point. This process is repeated for the remaining combinations of each network stage.

formally in (5.3) where $\oplus$ denotes the so-defined TAP combination operator.

$$f \oplus_{p,q,k} g : \mathbf{r} \mapsto f(\mathbf{r}_1) \cdot (1 - p_B) \quad (5.3)$$

$$\text{where } (\mathbf{r}_1, \mathbf{r}_2) = \arg \max_{\mathbf{r}_1, \mathbf{r}_2} f(\mathbf{r}_1) \cdot (1 - q_B)$$

Equation 5.3 expresses intuitively, that for a given total resource budget $\mathbf{r}$ we should divide the resources according to the values for $\mathbf{r}_1$ for the first stage and $\mathbf{r}_2$ for the second stage. This results in the maximal upper bound of the throughput of combined designs by finding the trade-off between maximising $f(\mathbf{r}_1)$ and minimising q_B , with respect to the design-time estimation of the exit probability distribution p .

To provide some insight into how parameters k , the first stage throughput, and the mean of the second stage throughput (weighted by p and $1 - p$) determine the blocking probability p_B , we detail the behaviour in the following section.

5.5 Finite Buffer Approximation Behaviour

The traffic intensity (ρ) is the ratio of the rates at which items enter and leave the buffer between the stages. These rates can be calculated from the initiation intervals of the design stages, and taken from the TAP curves of each stage. In the case of the second stage, the rate is the weighted mean of the early-exit II and the second stage II. Using the values from the TAP functions, ρ is calculated approximately in (5.4).

$$\rho \approx \frac{f(\mathbf{r}_1) \cdot p}{g(\mathbf{r}_2)} \quad (5.4)$$

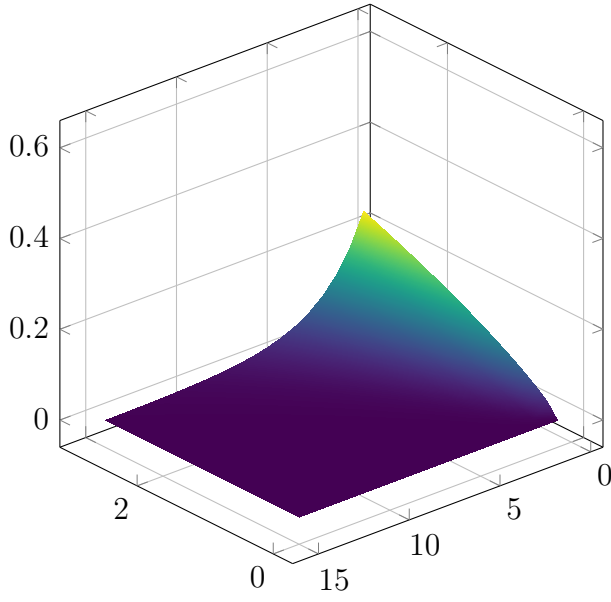
The coefficient of variation (s^2) can also be approximated due to the significant difference in the two service times of the second stage. What (5.5) highlights, is that the variance of the system is dominated by the probability of hard inputs rather than the incoming and outgoing

rates.

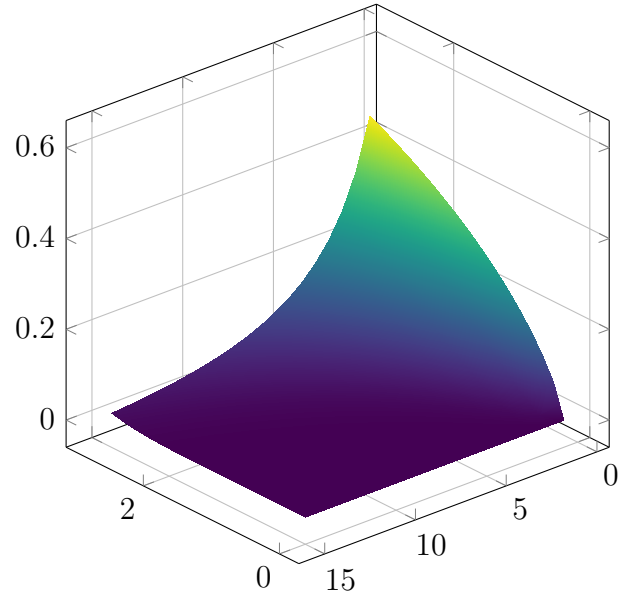
$$s^2 \approx \frac{1}{p} - 1 \quad (5.5)$$

When using fpgaConvNet to generate the FPGA mappings of the first and second stages, the resulting designs are not continuous along the Pareto front due to the discrete nature of the parameters of the template components. Therefore, ATHEENA will not be able to perfectly match the predicted throughputs to produce a traffic intensity $\rho = 1$. We provide plots of the behaviour of p_B in Figure 5.6 for a range of values: $\rho \in [0, 3]$, $k \in [1, 16]$.

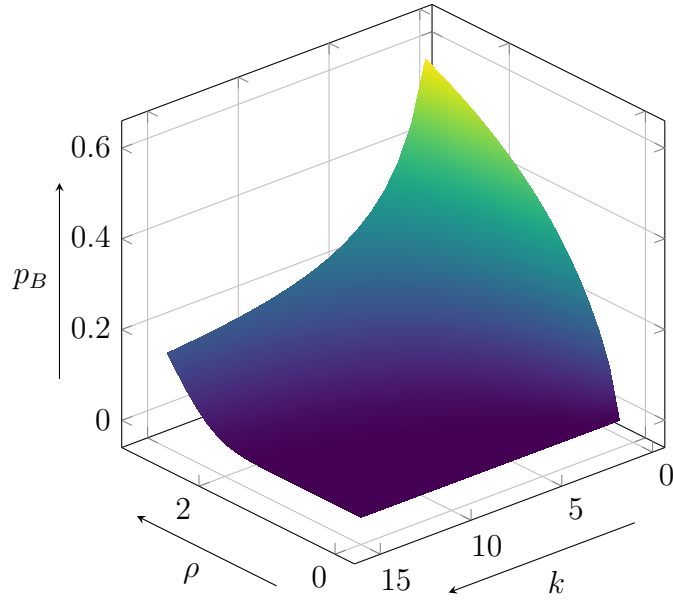
When $\rho < 1$, we consider this an over-provisioning of resources for the second stage meaning that, for a large enough queue, throughput is determined by the first stage. Conversely, when $\rho > 1$ the throughput of the accelerator is limited by the scaled throughput of the second stage. Under real-world conditions, the throughput of the full accelerator will fluctuate due to the stochastic nature of the second stage utilisation. This is also illustrated in the shaded region of Figure 5.7. In the case that the second stage is the limiting factor, a reduction in the use of this stage, corresponding to $q < p$, will result in an increase in throughput due to a reduction in traffic intensity and blocking probability. For $q > p$, the throughput will be reduced due to the reliance on the limiting stage. The solid purple line corresponds to $q = p$, in which the probability distribution of hard samples versus easy samples matches that of the profiled, design-time probability.



(5.6.1) $p = 0.10$



(5.6.2) $p = 0.25$



(5.6.3) $p = 0.40$

Figure 5.6: Blocking probability p_B against traffic intensity ρ and queue size k , for a fixed p .

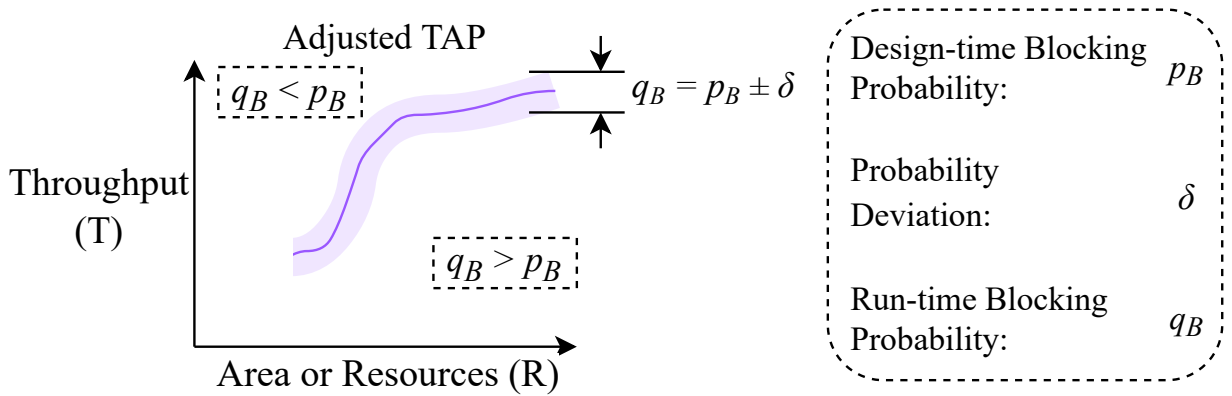


Figure 5.7: Estimation of the adjusted TAP function with respect the deviations (δ) in blocking probability at run-time (q_B) from design time (p_B).

5.6 Summary

This chapter addressed a weakness in the original methodology: the reliance on assuming sufficient buffer sizing. There are several approaches based on queueing theory that have a theoretical basis on which to build the throughput estimation. Through comparison against a simulated abstraction of the hardware, each approach was assessed in terms of its accuracy and computational efficiency. This resulted in choosing an approximate method that uses a general distribution, characterised by the mean and variance, to calculate the blocking probability of the first stage by the second. Therefore, using Queueing Theory to model the accelerator designs generated by ATHEENA, it is possible to include the actual queue size in determining the throughput of a given combination of design stages. By re-framing the resource allocation in terms of this approximation we could then explore the behaviour of the system, highlighting the effect the rates and queue size have on the blocking probability and by extension, the throughput of the full accelerator system. Using the new methodology, we address the improvements in the benchmarking results from Chapter 4 as well as providing further evidence for the necessity of the finite buffer model in the performance of EE-CNN accelerators.

CHAPTER

6

Improved Resource and Throughput Estimation for Early-Exit CNN Accelerators on FPGAs

The following chapter details the modifications to the toolflow to improve the resource modelling. We then explore the accelerator designs generated by the toolflow with the improved resource modelling and throughput estimation based on Queueing Theory. The included results are from the journal extension submitted to IEEE TCAD.

6.1 Introduction

The aim of this chapter is to demonstrate the benefits of the Queueing Theory modelling approach from Chapter 5. In conjunction with updated resource modelling, the Queueing Theory throughput estimation method is integrated directly into the existing ATHEENA toolflow in place of the simple estimation function used previously in Chapters 3 & 4. Having shown that the throughput estimation has increased accuracy, we construct scenarios in which a better understanding of the throughput, in relation to the resource allocation and queue size, is neces-

sary to prevent significant performance degradation in the designs generated through DSE. We then return to the benchmark results of Chapter 4 in order to establish the effect of the new resource and throughput estimation on the accelerator designs implemented on an FPGA. Separate to the benefits of accurate throughput and resource modelling, we address the notion of buffer *placement* within the accelerator design. We carried out some preliminary experiments on case-study networks to establish the effect of moving the buffer to after the immediate branching point. For the case-study network, we found that the original placement had the greatest performance. In the interest of future work, we detail potential cases in which this optimisation could be utilised to enhance performance beyond the gains of accurate throughput and resource estimation.

6.2 Toolflow Improvements

Once again, the ATHEENA toolflow is extending the existing, open-source, fpgaConvNet toolflow. Further changes and additions are highlighted in Figure 6.1 as blue and red respectively. The main new component added to the flow following Chapter 5 is the Queueing Theory-based approximation method for throughput estimation. The improved resource modelling is built into the DSE through the module and layer resource estimation functions.

6.2.1 Updated Resource Modelling

The baseline resource model estimates the values of each FPGA resource consumed by an accelerator design using statistics gathered from a database of pre-compiled and implemented components. The collection of modules and layers have randomly generated configurations. Each version is compiled with Vivado HLS and implemented in Vivado IP integrator. The results of the LUT and FF consumption are recorded and a simple regression model is used to determine the mapping from configuration to resource consumption. The database is constructed of real-

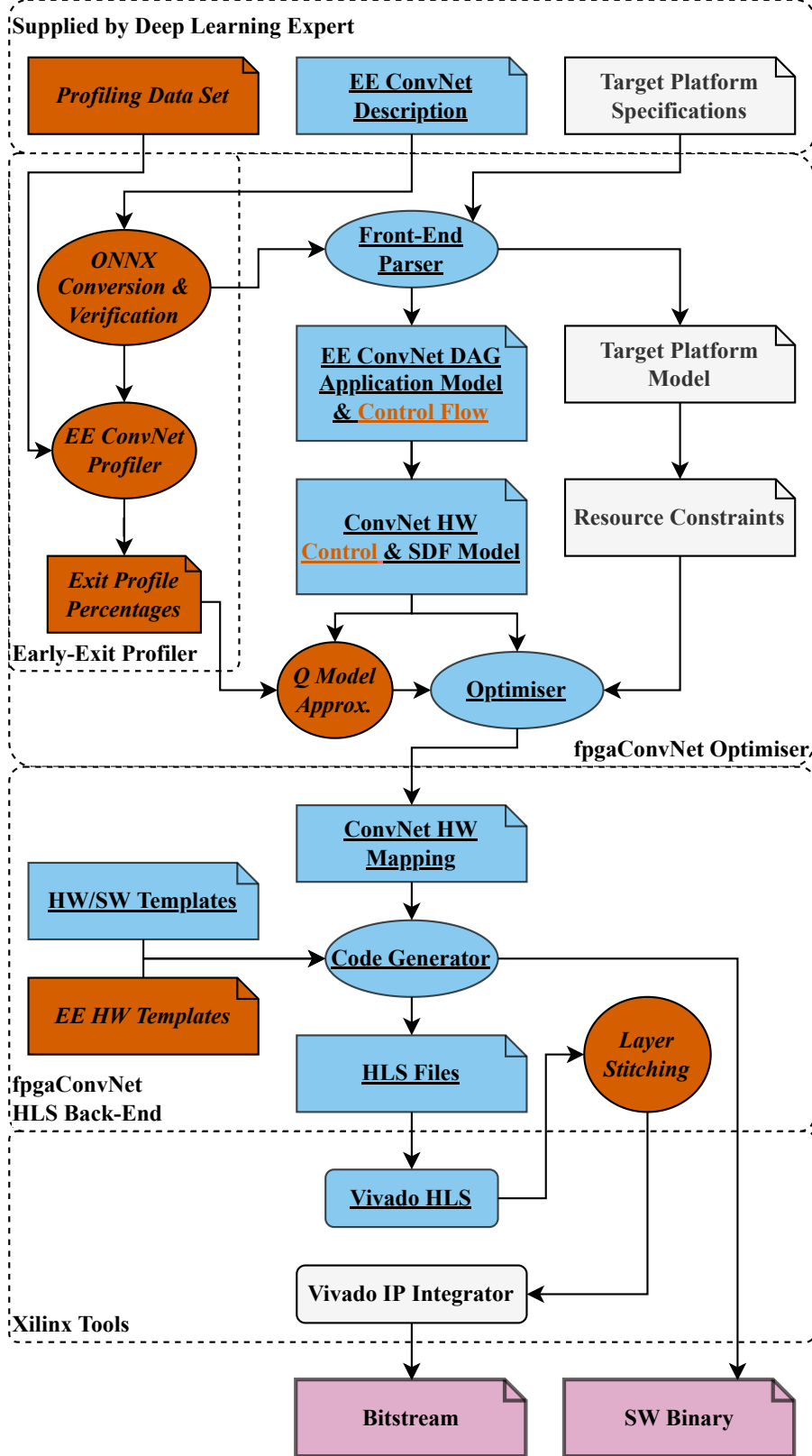


Figure 6.1: A visual representation of the improved ATHEENA toolflow. We have built upon the baseline, open-source fpgaConvNet toolflow. New elements are in orange italicised and modified elements are in blue and underlined.

istic valued configurations in order to provide an accurate estimate of the resources consumed. Since there are changes to the HLS templates and additional components, we compile our own set of statistics to inform resource estimation. The changes to the resource model allow us to maintain the accuracy of the LUT and FF estimation. The DSP model required no changes as there is an existing explicit formula for their estimation used by the convolution and linear layers. The BRAM model estimation requires heuristics that depend on the width and depth of a given storage component as well as the target device, and the amount of FIFO-based storage automatically inserted at the sub-component interfaces. Older devices, such as those of the Xilinx 7 series, require the number of discrete BRAMs to be a power of two. We add this behaviour into our cost model to improve the estimation accuracy for these devices as well as properly factoring in the discrete sizes of BRAM-based memory components in the accelerator DSE. These changes resulted in an improvement in the resource modelling as well as being able to maximise the utilisation of the BRAMs by keeping buffer sizes within a power of two where appropriate.

6.3 Experimental Results

The first experiment aims to show the difference in performance that Queueing Theory throughput estimation in resource-constrained scenarios that have the potential to occur in a deployment environment. The second experiment returns to the case-study network to demonstrate the positive effect of improving both the resource and throughput model. Finally, we address the benchmarks networks from Chapter 4 to verify performance improvements on other CNN models.

6.3.1 Effect of Queueing Theory

To demonstrate the performance difference of the Queueing Theory-based throughput modelling, separate from the improved resource modelling, we investigated a different combination of network and FPGA device (Tiny ImageNet-scale [94] B-LeNet and ZCU104). The Tiny ImageNet-scale input is double the height and width of the original B-LeNet network input, with the subsequent layers having the same depth but increased width. The network was chosen for the large input size in conjunction with the FPGA device having a limited number of BRAMs. This combination of network and device increases the pressure on the limited resource by reducing the BRAMs available for the finite buffer and making the intermediate feature maps larger, requiring more BRAMs for a given queue size. A set of first and second network stages were generated using the optimiser before being combined in the following approaches:

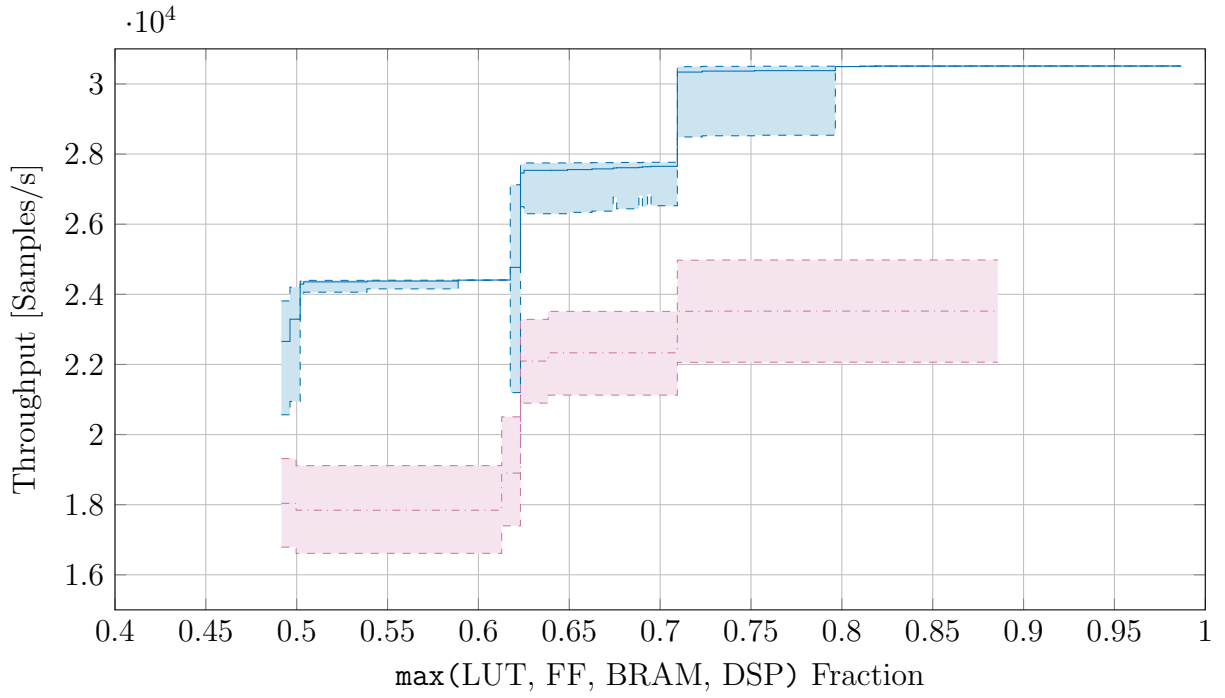
1. The remaining BRAM was added to the conditional buffer in proportion with the other resources, and the stages were combined under the Queueing Theory methodology detailed in Chapter 5.
2. No extra BRAM added to the conditional buffer beyond the expected minimum queue size of two, and the stages were combined *without* the use of the Queueing Theory throughput model.
3. The remaining available BRAM was added to the conditional buffer in proportion with the other resources, and the stages were combined *without* the use of the Queueing Theory throughput model.

The approaches are compared in Figure 6.2. Each solid line represents the ideal situation in which the design-time exit probability matches the run-time probability. The lines are bounded by a $q = p \pm 5\%$ deviation in run-time probability. In both parts of the figure, the blue solid curves represent the Pareto front of designs where the network stages are combined with the

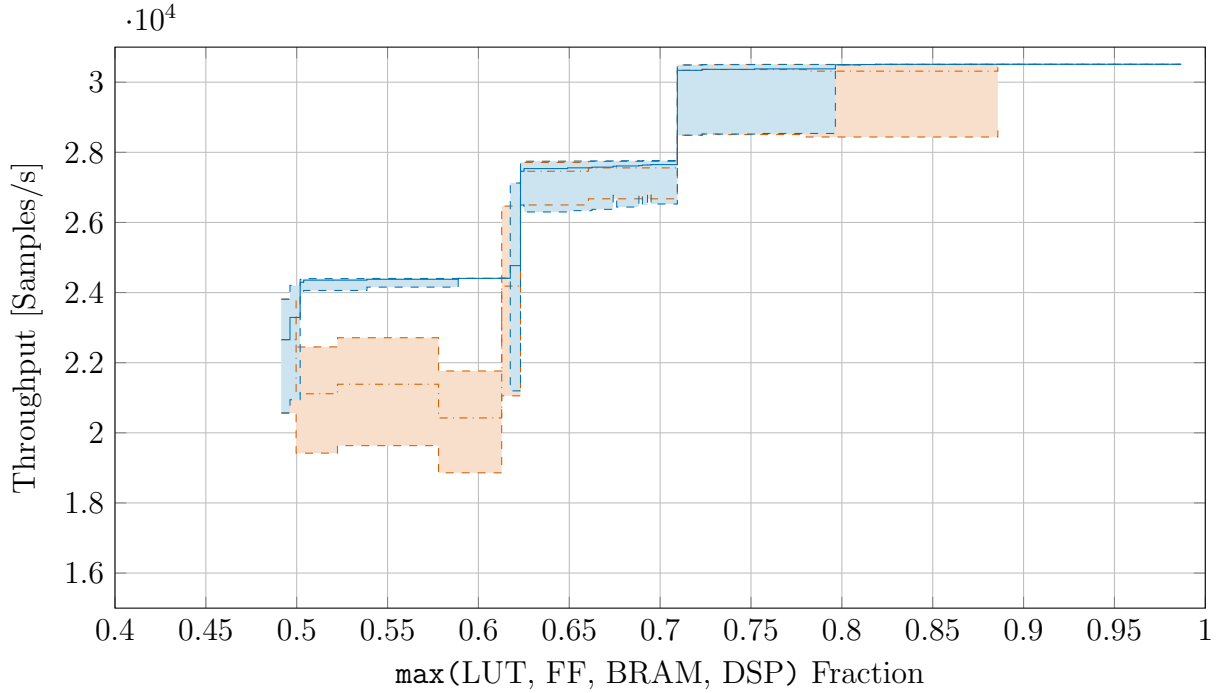
throughput estimate informed by Queueing Theory. The yellow and red/pink dashdot curves represent the Pareto front of designs combined with the old method of throughput estimation, however, the points plotted have had their throughput values recalculated using Queueing Theory to provide the most accurate estimation.

The main takeaway from this experiment is that, in all cases, the modelled throughput achieved by the toolflow using the new throughput estimation matches or exceeds the other methods of stage combination. This is most obvious in comparison with the red/pink dashdot curve in Figure 6.2.1. This scenario represents a device environment in which the BRAM is highly constrained. This is not an uncommon occurrence in accelerator design as we cannot always assume we have unlimited access to the FPGA’s resources due to other IP cores or additional processing system components that are not included in our resource estimations. We can see that combining the network stages according the throughput derived from queueing theory avoids a $0.73 - 0.80\times$ degradation in throughput. Furthermore, the gap between the red/pink dashdot and blue solid curve illustrates the limiting effect of a low queue size. The designs bordering the red/pink dashdot curve are particularly vulnerable to this case as the resources will be prioritised for use by the computational part of the first and second stage, rather than the finite buffers. This also increases the variability across different exit probabilities at run-time. Utilising the available BRAM to augment the queue size is important in producing reliable run-time performance.

The main reason for the difference in throughput between the blue and red/pink curves is the limitation of the queue size in the designs of the red curve. The low queue size drastically increases the blocking probability p_B of the second stage, meaning that a heavily constrained system (with respect to BRAM) will not be able to benefit as much from the ATHEENA toolflow. A key strength of the new throughput model is that now we are able see, at the design stage, the effect of adversarial environments on the accelerator.



(6.2.1) The pink line represents the scenario of a BRAM-constrained deployment environment.



(6.2.2) The yellow line represents a deployment environment where BRAMs are limited proportionally to other resources.

Figure 6.2: Comparison of modelled performance between the Queueing Theory-enabled toolflow and the original method for combining network stages. We have plotted the results of the three scenarios detailed at the head of Section 6.3.1. The results of scenario 1 are depicted by the blue line, which uses Queueing Theory to predict throughput. The outcome of scenario 2 is represented by the red/pink dashdot line, in which the minimum the queue size is locked to 2 during `fpgaConvNet` optimisation step. Scenario 3 is shown in the orange/yellow dashdot line and has the same `fpgaConvNet` optimisation as scenario 1. Neither scenario 2 or 3 use Queueing Theory to choose the combination of network stages. However, each point plotted on the graph has had its throughput accurately computed using Queueing Theory in order to compare the TAP functions.

Comparing the blue solid and yellow dashdot curve in Figure 6.2.2 demonstrates that, even in the presence of available BRAM, a method informed by Queueing Theory can avoid a throughput degradation of $0.84\times$. We can see from the shaded regions that, between the 0.5 and 0.6 resource fractions that the blue solid curve is more robust to perturbations in run-time throughput. Towards the higher resource fractions the gap becomes less pronounced as the proportion of BRAMs required for a sufficient queue size is lower, relative to the BRAM required by the network stages for computation.

At the 0.8 resource fraction we see a prime example of the original tool selecting what appears to be a better combination of network stages, however, the queue informed tool factors in the throughput limitation from a smaller queue size and selects a more robust combination of stages. This highlights the fact that, even when design time throughput is nominally the same, the variation at run-time can be mitigated using the Queueing Theory informed estimation method.

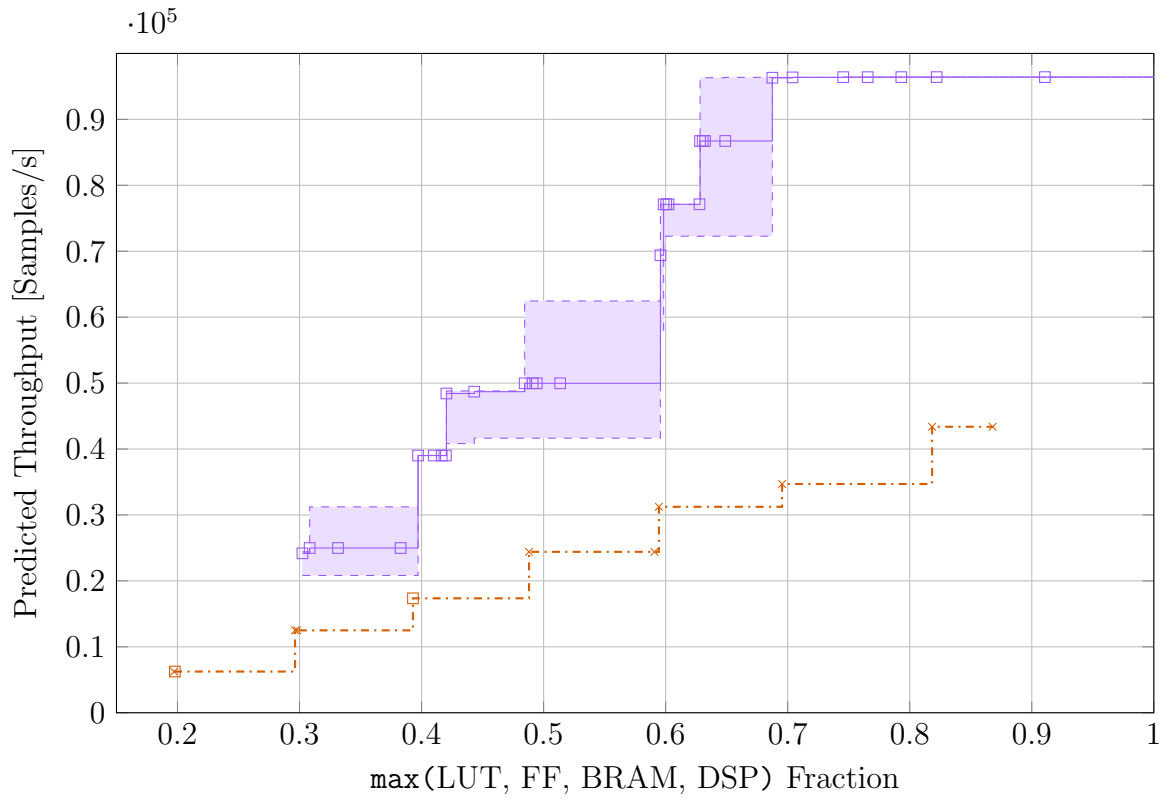
6.3.2 Returning to BranchyNet Case-Study

The minor modifications to the case-study network detailed in Section 4.5 and Figure 4.4 have been retained for use with the new toolflow. We continue to use the experimental setup that follows fpgaConvNet [25], comparing to the same baseline. The target device used is the Xilinx ZC706 board with the small Zynq 7045 FPGA SoC, however, the FPGA has proportionally more BRAM blocks than the ZCU104 used in the previous experiment. The resources available are 218,600 LUTs, 437,200 FFs, 900 DSPs, and 1,090 18K BRAMs. The accelerator designs generated by ATHEENA are compared to a corresponding single-stage network baseline. This single-stage baseline consists of the network layers from the start of the Early-Exit network through to the end of the second stage. To generate a Throughput-Area Pareto *curve*, we constrain the ATHEENA and baseline optimisers with stepped fractions of the device’s resources.

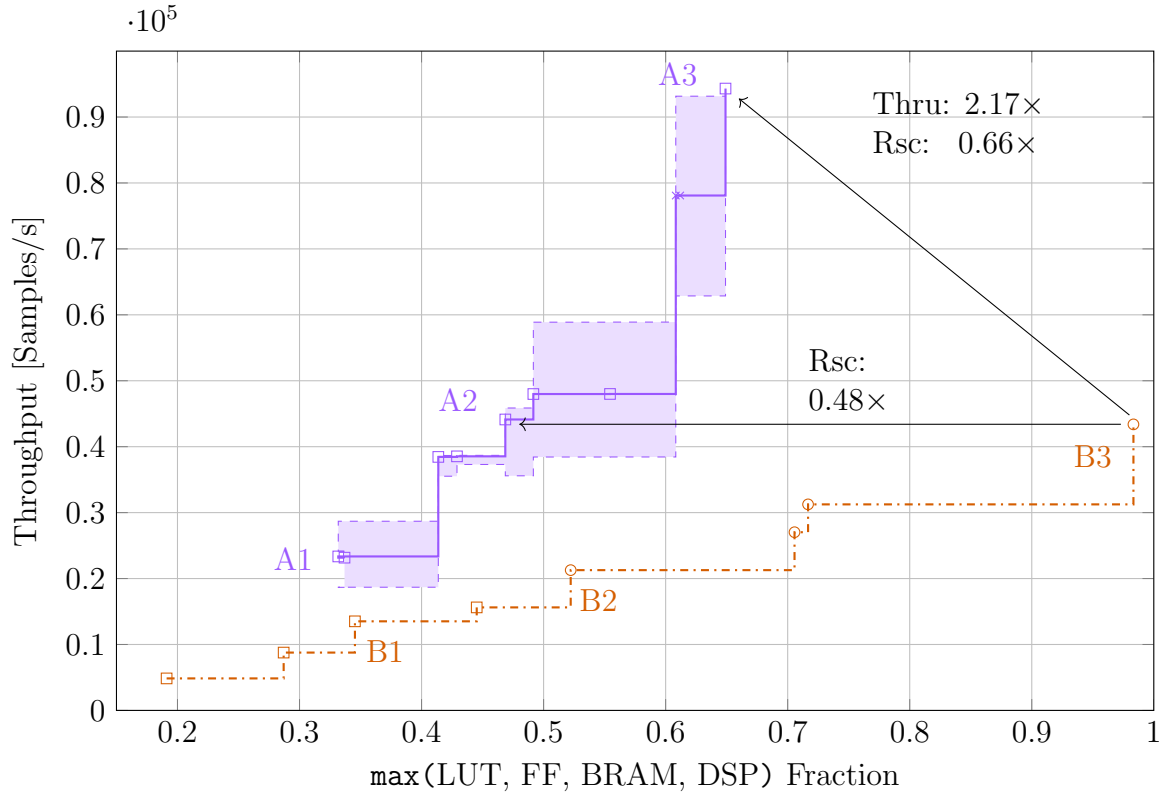
Due to randomisation in the optimisers, they are run ten times and the best points are chosen to form the curve. The set of generated hardware mappings for each stage are collated for the new modelling approach in order to produce a Pareto front of the best performing accelerator designs for a given fraction of the target device’s resources. These predicted results are shown in Figure 6.3.1. A subset of these results are then used to generate C++ code for compilation with Vivado HLS and implementation in order to produce bitstreams for evaluation against the baseline and predicted designs. Figure 6.3.2 shows the resulting Throughput-Area Pareto curve after synthesis, implementation and place-and-route. The automatically generated host code for the board loads a batch of 1024 samples onto off-chip memory and enables the DMA transfers to and from the design. We measure the total time taken from the start of the DMA transfer until the DMA block registers as being idle and use this to calculate the throughput of the network.

The orange dashdot lines denote the baseline results in both parts of Figure 6.3. While the open-source fpgaConvNet model is not accurate on a point by point basis, the trend is consistent for the predicted and board implementations. For higher resource allowances, the designs tend towards being limited by DSPs, this is due to the baseline optimiser selecting an increased level of parallelism for the Convolution and Linear layers in addition to the FPGA having a lower proportion of hardened DSP blocks. ATHEENA benefits from the limited DSPs of the device as the second network stage (containing convolution and linear layers) has a reduced throughput requirement. This results in the upper limit of the implemented designs being constrained by LUT utilisation.

The solid purple line shows the case when the q (percentage of easy samples in the test set) matches that of p (the profiling data set percentage), in this case $q = p = 25\%$. The dashed lines represent a range of points taken on the Throughput-Area space with a differing percentage of hard samples. In the predicted results, this is done by adjusting the characteristics of the



(6.3.1) Throughput Area results generated by the simulated annealing optimiser stage of the toolflow at a p of 25% and $q = p \pm 5\%$.



(6.3.2) Throughput Area results obtained using ZC706 board. At a p of 25%. Tested using data sets consisting of $q = 30\%, 25\%, 20\%$.

Figure 6.3: Baseline results for the open-source HLS-based *fpgaConvNet* are red and the limiting resources for each point are denoted by: $\times$ (BRAM), $\square$ (LUT), $\circ$ (DSP). Designs A1, A2, A3, and B1, B2, B3 have detailed resource usage in Table 6.1.

two-point distribution that informs the Queueing Theory approximation. For the implemented results on the device, a random batch of samples is selected from the test data with a proportion of ‘easy’ and ‘hard’ samples that match the the required test probabilities. The lower dashed line represents a $q = 30\%$ and demonstrates a partially reduced throughput for some of the data points. The upper dashed line represents a $q = 20\%$ and highlights that some sub-optimal profiling situations can still result in throughput increase thanks to the significant latency reduction of data samples exiting from the high-throughput first stage.

Ultimately, the results demonstrate the improved throughput with run-time cases that may exhibit a slight decrease in throughput from variation in exit distribution within a batch. While there is not a large difference, with respect to the on-device results, when compared to the ATHEENA results from Section 4.5, the lower resource utilisation benefits from a throughput increase with negligible additional resource consumption. The best performing design points produced with the new approach reduces the resource consumption by 16 percentage points from the previous best performing point. This is a product of both the resource and throughput modelling providing more accurate information for the DSE.

These improvements in results stem from the improved modelling of both the throughput and resources. The improved resource model means that the number of design points in which the resources are under or over-estimate are reduced. Therefore, when the stages are combined, the resulting points on the Pareto front are correctly positioned on the x -axis. The Queueing Theory-based throughput estimation is much more accurate with the inclusion of the finite buffer behaviour, causing the Pareto-optimal design points to be correctly positioned on the y -axis. This improves the resolution of the combined TAP function, allowing the model to determine specific Pareto-optimal points as opposed to approximate points following a general trend. Quantitatively, we note that the average deviation in throughput and resource consumption is now only 5.3% and 5.8% respectively. These remaining deviations are likely due to the

Table 6.1: Tabulated resource comparison for implemented Baseline Vs ATHEENA on ZC706 board. *Maximum limiting resource.

	LUT	FF	DSP	BRAM	Limiting Resource (%)	Throughput (Samples/s)
B1	75513*	61361	295	55	35	13513
A1	68383*	63170	169	206	31	19434
B2	105451	84761	470*	89	52	21276
A2	128940*	117138	407	239	59	47583
B3	138194	120764	885*	170	98	43384
A3	163637	142913	742*	265	82	94170

variability of HLS compilation and module-level latency estimation. Finally, the gap between lower dashed line and the baseline continues to indicate the robustness of the approach to the difference between p and q for a real-world application.

The maximum measured ATHEENA throughput is $2.17\times$ the maximum measured baseline throughput. ATHEENA achieves this throughput using 66% of the resources when apportioned between the first and second stage according to the profiled probabilities. Alternatively, ATHEENA can achieve the same throughput as the maximum baseline using 48% of the design’s limiting resource. This is a marginal 2 percentage point increase when compared to the results from Section 4.5 so can likely be explained by variation in the optimisation approach that generates the network stages. The design points in Table 4.1 show that the achieved throughput increase comes at the cost of an increase in BRAM which confirms the necessity of providing sufficient memory resources as well as modelling the impact on throughput when this is not possible. The designs A1, A2, and A3 (labelled in Figure 6.3.2) require this as part of the conditional buffers.

6.3.3 Benchmarking Results

We include an additional two Early-Exit networks in Table 6.2. The table shows the best performing predicted throughput and corresponding resource results from both the optimizer stage of the baseline and ATHEENA toolflows. Due to the increased size of these networks

Table 6.2: Resulting throughput improvement for two-stage accelerator designs generated by ATHEENA model compared to *fpgaConvNet* baseline. *Implemented on Xilinx ZC706

Network (Task)	Toolflow	Limiting Resource		p (%)	Throughput	
		Type	%		(Samples/s)	Gain
B-LeNet[22] (MNIST[88])	Baseline*	DSP	84	-	43384	1.00×
	ATHEENA*	LUT	65	25	94317	2.17×
Triple Wins[91] (MNIST[88])	Baseline	DSP	86	-	19524	1.00×
	ATHEENA	DSP	99	25	60988	3.12×
B-AlexNet[22] (CIFAR10[92])	Baseline	DSP	84	-	8676	1.00×
	ATHEENA	DSP	94	34	17357	2.00×

we target the Xilinx VU440. The resource proportions of this device are such that the relative amount of DSPs are lower and there is a higher proportion of LUTs. This does appear to affect the limiting resource in a predictable way as the limiting resource for the larger networks is DSPs. We use the percentage of ‘hard’ inputs outlined in their respective papers to generate the two-stage design. Utilising the improved ATHEENA results in a throughput gain of 2.00× – 3.12× at the cost of increased resources. In comparison to same benchmark networks in Table 4.4, the benefits of the new approach are not as clear cut as the original. We have the obvious benefit in the reduction of resources used in the BranchyNet case-study. In contrast, the B-AlexNet network requires a resource increase of 6 percentage points to match the throughput improvement of 2.00× relative to the baseline. The Triple Wins network does benefit from an increase in performance in which the relative throughput improvement goes from 2.78× to 3.12×. However, this does incur an additional 18 percentage points of resource consumption, potentially limiting the implementation on a real-world device.

6.4 Addressing Buffer Placement

In this chapter we evaluated the ATHEENA toolflow with respect to the developments in modelling resource consumption and increasing the accuracy of throughput estimation using Queueing Theory. This illustrated the importance of finite buffer *sizing* but fails to address

the placement of the buffer component in relation to the computational stages of the network design. The accelerator is constructed of a series of deeply pipelined, synchronous components coupled with streaming interfaces to transfer data. Additionally, the transparency of the buffer with respect to the computational components of the network indicates that the buffer can theoretically be placed at any point along the backbone network stage without affecting the validity of the classification results.

6.4.1 Benefits of Alternative Placement

The placement of buffer components within FPGA-accelerated networks influences both the performance and efficiency of the system. The main benefit of strategic buffer placement is the reduction of the latency *difference* between the storage of intermediate feature maps and the production of decision results. This latency difference stems from the additional branch computation between the intermediate feature map storage and Early-Exit decision component. By migrating the buffer to a deeper point in the backbone, this latency difference has the potential to be eliminated, reducing the minimum required depth of the finite buffer for a functional accelerator. The second benefit is related to the downsampling structure of standard CNN architectures. In general, the dimensions of the intermediate feature map are decreased deeper in the backbone. This reduces the BRAM blocks required per intermediate feature map stored in the queue.

6.4.2 Disadvantages of Altering Placement

The primary reason for the improved performance of the Early-Exit accelerator compared to the baseline is the ability of the toolflow to efficiently apportion resources between network stages with differing throughput requirements. The buffer placement defines the boundary between these regions, meaning that, as you migrate the buffer, the components that were in

the lower throughput region now require a significantly higher throughput. In the BranchyNet case, the throughput needs to be quadrupled, drastically increasing the component’s resource consumption.

6.4.3 Buffer Placement Experiment

Initial experiments conducted during the development of the improved resource and throughput modelling revealed insights regarding buffer placement. These were carried out on the BranchyNet case-study network B-LeNet. An exhaustive search of placement points on the backbone was conducted, followed by the optimisation of subsequent combinations of the, now altered, design stages. This resulted in a set of Pareto curves, like those in Figure 6.3, that correspond to each different buffer placement for this example network. Comparing these, it was clear that the best performing designs belonged to the original Pareto front for the initial buffer placement. This suggests that while the BRAM consumption can be dramatically reduced by moving the buffer, the precise placement results in an overall increase in resource consumption for a given throughput. This is exaggerated in the case-study network as the convolution layer at the border between the different throughput regions cannot be fully parallelised on the device, creating a bottleneck and significantly reducing the overall throughput of the accelerator.

While the initial experiments did not display benefits, some examples of scenarios in which buffer placement is particularly important are as follows. When the on-chip memory is limited relative to other FPGA resources, such as on Kintex Ultrascale+ devices, which have a 2:1 ratio of DSPs to BRAMs [99]. Additionally, as networks grow deeper, Early-Exit branch placement is more likely to coincide with narrower layers due to the progressive downsampling and reduction in computation seen in efficient CNN architectures like MobileNet [19] and EfficientNet [20]. Strategic buffer placement in these cases can optimise resource utilisation and enhance overall

performance.

6.5 Summary

To efficiently explore the design space for our Early-Exit network accelerators we must estimate the resources and throughput for many points to generate a front of Pareto optimal designs. We accomplished this using abstraction and approximation techniques taken from Queueing Theory. These abstractions allowed us to factor in the realistic *constrained* buffer sizes while the approximations prevent excessive computation time and maintain accuracy. This culminates in the development of a toolflow that enables the DSE of the Throughput-Area Pareto-optimal hardware implementations of Early-Exit networks. This chapter evaluated the new approach to throughput estimation through experiments designed to limit the BRAM resources necessary for the finite buffer to operate effectively. We then returned to the BranchyNet case-study and benchmark networks to re-evaluate the toolflow in this instance, observing some significant throughput increases or resource reductions. Finally, we addressed the issue of finite buffer *placement* as a potential route for further optimisation of Early-Exit networks on FPGAs.

CHAPTER

7

Improving the Early-Exit Decision Component and Corresponding Hardware

The development work in this chapter was completed with Luigi Rinaldi as part of my supervising an Undergraduate Research OPportunity. The goal of this work is to improve upon the initial decision component, reducing resource consumption and latency, whilst retaining the efficacy as a confidence metric in discriminating inputs of varying difficulty.

7.1 Introduction

First discussed in Section 4.3.1, the initial implementation of the Softmax Decision component was focused on closely matching the behaviour of the equivalent operations carried out in software on a CPU/GPU. The resulting hardware relies on evaluating the exponential function on each element of the branch output. On an FPGA, this computation is expensive, both in terms of resources and latency. These issues are worsened in the initial design due to the use of floating point IP cores in order to preserve the range of the Softmax function at the bounds

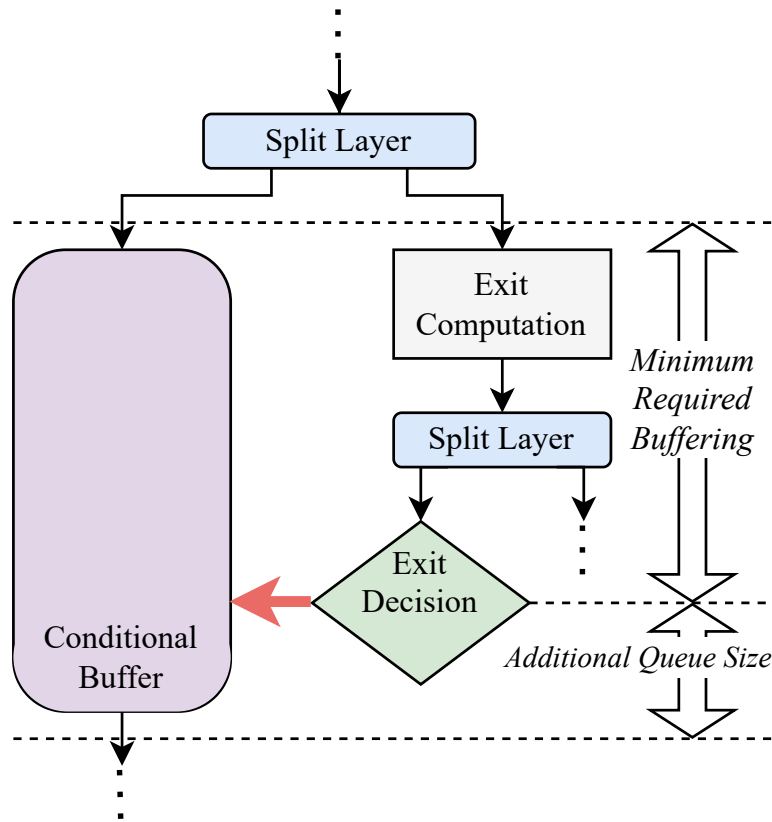


Figure 7.1: The latency of the additional exit computation and exit decision layers is important in deciding the minimum amount of buffering required by the conditional buffer to prevent deadlock in the design.

of the fixed point input. The use of floating point cores is continued in both the summation, reduction, and comparison modules, further increasing the latency and resource consumption of the component overall. This has an impact on the rest of the accelerator, illustrated in Figure 7.1, by requiring an increase in the buffer size. It is by addressing these challenges that we aim to reduce the overhead of the Early-Exit hardware, both in terms of resources consumed and additional latency in the confidence calculation.

The approach detailed in this chapter is a result of tackling the problems from the machine learning *and* hardware design directions. We ask ourselves: why is the Softmax function used? Can we replicate the behaviour with alternative functions? What are the implications of changing the component’s precision on the assessment of confidence? How can we best navigate the balance between a faithful representation of Early-Exit behaviour and optimising for an FPGA implementation? As discussed in Section 2.1.1, an issue with *ad hoc* confidence metrics is the

tendency of DNNs to exhibit overconfidence on inputs dissimilar to the training data, leading to the network being confident in incorrect results. One method to address this is *learned* confidence metrics [46]. These can improve accuracy and limit overconfidence. Unfortunately, the complexity of the hardware implementation and the significant impact on training make them unsuitable for our purposes. We instead opt for modifications to the current confidence metric that better target FPGA hardware. While these alterations go some way toward addressing the resource and latency issues, we also provide insights into the confidence differences on an example network and test inputs. To this end, we explore the impact of numerical base on the Max Softmax (top1) confidence metric and evaluate the changes in difficulty classification with an example Early-Exit CNN. The results of this investigation inform the choices for the decision component hardware. Our evaluation then addresses the performance of the chosen confidence metric (coupled with a trained Early-Exit CNN) as a discriminator for ‘easy’ and ‘hard’ samples. Initially, we look at the distribution of values produced by the confidence metric in order to highlight any significant differences. We then take measures of each metric that draw on the evaluation approaches for threshold-based binary classification, specifically we look at the Area Under the Receiver Operating Characteristic (AUROC) [100], and the Area Under the Precision-Recall Curve (AUPRC) [101], [102]. These measures segregate the metric results into four categories: true positive, false positive, true negative, and false negative [103]. We define these categories based on whether or not we think the input should have exited early. Using these measures, in conjunction with exit rate, the similarities between the metrics can be highlighted. Finally, we provide a quantitative view of the resource and latency (post synthesis and implementation) for the original metrics and our chosen metric.

7.2 Improving the Decision Component

To address the issues with the original decision component we first take a look at its construction. The first version of the component was designed to compute the result of (7.1) in which C is the number of classes, and x_i are the raw values produced by the classification layers of the Early-Exit CNN. The computation is divided amongst four modules, as depicted in Figure 7.2.

$$\max_{i \in \{1 \dots C\}} \exp(x_i) > C_{thr} \cdot \sum_{j=1}^C \exp(x_j) \quad (7.1)$$

The first of these modules takes the stream of raw values in the form of an arbitrary-precision fixed-point number with a range of $[-128, 128)$. Using pre-written HLS IP cores for `fixedToFloat` conversion and `exp` functions, these values are transformed into single-precision floating-point numbers and exponentiated. These values are then streamed to the `ReduceMax` and `Sum` modules in parallel to compute the maximum value and the sum over all class values. The original design uses a simple tree structure to try and mitigate the latency introduced by the floating-point comparison and addition. Lastly, the `compare` module multiplies the incoming sum by the user-defined threshold and determines whether or not the maximum value exceeds that result.

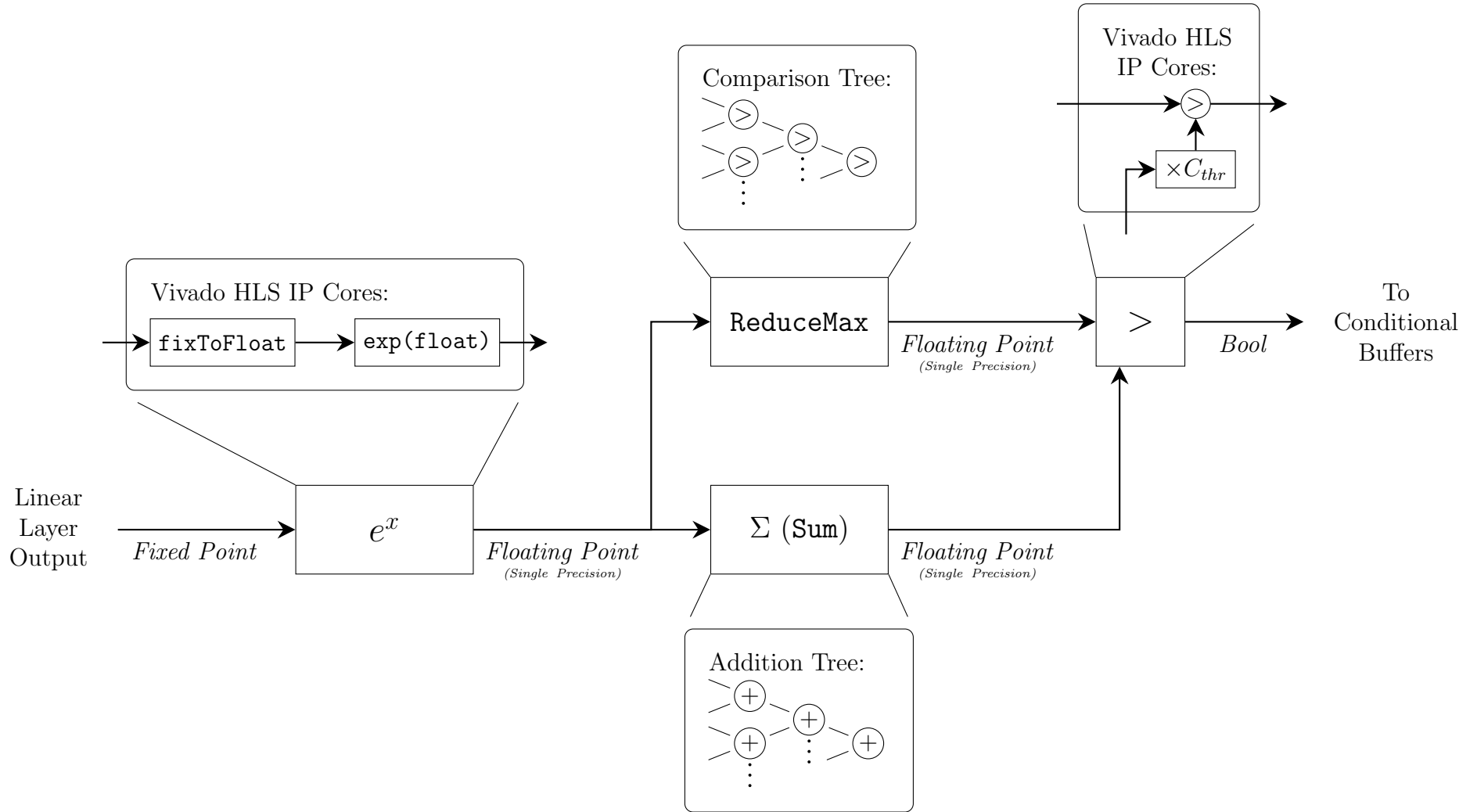


Figure 7.2: Block diagram of the Softmax Decision Component comprising of design-time configurable modules.

The initial version of this component was designed prioritising accuracy with respect to the PyTorch-based top1 Softmax metric used to evaluate the Early-Exit CNNs on standard CPU/GPU hardware. The second priority was throughput (at the potential cost of latency and resources). This meant increasing pipeline depth and parallelism. Unfortunately, in networks with significant branch latency, the pipeline depth of the decision component compounds the branch latency. This is due to the Vivado HLS pipeline requiring several sets of samples to be processed before outputting a result. To mitigate this, we removed the pipelining in the floating-point components, increasing the initiation interval of the `exp`, comparison, and summation modules. The effect of this is a modest reduction in resources and a significant increase in per-input latency for the stand-alone component. In a full accelerator system, this has the effect of *reducing* the overall branch latency as the decision component’s increased latency is masked by that of the convolutional layers in the branch. Both of these original versions have been included in the evaluation in Section 7.3.4.

FPGAs have become increasingly more equipped with dedicated hardened blocks capable of performing floating point operations efficiently. However, the floating-point operations in the original decision component are both latency and resource intensive. Therefore, in our resource-constrained environment, the new design needs to avoid multi-cycle, floating-point calculations.

The structure of the new FPGA hardware for the improved Decision Component is now condensed into a single module with the C++ code for HLS. This module performs each operation sequentially in a pipelined fashion. However, the primary difference is that our new component stays in a fixed-point representation *throughout* the confidence calculation. We adjust the precision within the different modules to maximise the range represented. The second key difference is a change of base. The Softmax function is also known as the normalised exponential function and existing experimental results show that it is an effective measure of

confidence [23]. Unfortunately, the `exp` function requires multiple DSP blocks and cycles to compute only an approximate value for use in the decision component. In fixed-point hardware, 2^x (where $x \in \mathbb{Z}$) can be computed simply by shifting 1 to the left or right by the value of x . Replacing the `exp` operation from (7.1) with 2^x results in (7.2).

$$\max_{i \in \{1 \dots C\}} 2^{\lfloor x_i \rfloor} > C_{thr} \cdot \sum_{j=1}^C 2^{\lfloor x_j \rfloor} \quad (7.2)$$

Using 2^x in conjunction with truncating the values below the binary point does reduce resource usage and latency and, as we will see in the results section, has a negligible effect on the power of the metric as a easy/hard input discriminator. In the interest of preserving accuracy and precision, we subtract the maximum value of an input sample from each raw value ($x'_i = x_i - \max_{i \in \{1 \dots C\}}(x_i)$) before being fed into (7.2). This reduces each of the raw values to less than or equal to one, allowing us to capture some of the lost range from using a fixed-point precision value. Furthermore, the maximum value will always be 1, and all other values calculated for 2^x become less than one so can simply be right-shifted transformations of 1. Through an inexpensive comparison check with 0, we can further eliminate computation for cases in which $x = 0$ since $2^0 = 1$.

The changes made so far are primarily motivated by tailoring the confidence metric for hardware acceleration. The resulting changes introduce further quantisation error in the confidence metric. Furthermore, the reduction in base value means that it takes a greater difference between input values to achieve the same level of separation in the output values (as we will see in Section 7.3.1). It is possible to get an improved level of discrimination between ‘easy’ and ‘hard’ input samples but our results indicate it requires a different approach, such as training a separate discriminator or increasing base value. The evaluation aim is to demonstrate that the accuracy and precision matches that of the original confidence metric, allowing us to select an FPGA-amenable hardware implementation without fear of decreasing network performance.

7.3 New Decision Component Evaluation

The BranchyLeNet network trained on the MNIST [88] dataset detailed in Chapters 4 and 6 had a high test accuracy for its early classifier and final classifier meaning that there is limited variation in exit rate across threshold values. Since this variation is key in visualising the difference in confidence metrics, we train a slightly wider BranchyLeNet model on the CIFAR10 dataset [92]. This network was trained in the same fashion as the original BranchyNet paper: using a two stage process to first train the backbone, followed by jointly training the backbone and the early exit.

Due to the low capacity of the network, the top1 test accuracy of each exit was 60.80% and 61.24% respectively. To establish a baseline accuracy for the Early-Exit CNN, we determined the optimal thresholds for each confidence metric using the training data. These thresholds were used when determining the networks baseline accuracy under Early-Exit operation. The accuracy of the Early-Exit CNN with each metric marginally outperformed either exit individually, achieving between 62.65% and 63.33%. This highlights one of the benefits of the input-dependent nature of Early-Exit CNNs in avoiding ‘overthinking’. While not used in our accelerator, we include details for the information entropy-based confidence metric in our evaluation for completeness. In all evaluations, this metric performed worse than the top1-based metrics including a lower test accuracy of 62.62%.

7.3.1 Examining Raw Maximum Value Distributions

To provide insight into the problem faced by the confidence metrics, we include a histogram of the raw maximum values produced by the Early-Exit CNN *before* determining confidence. To produce these plots, the network was fed with the CIFAR10 test dataset and the raw values $x_{s,c}$ for each of the ten classes collected for every input sample. For the B-LeNet-CIFAR network, $s \in [1 \dots S]$ (where $S = 10,000$) is the number of input samples and $c \in C$ (where

$C = 10$) is the number of classes. We determine the maximum value of each sample’s raw values $\max_{c \in \{1 \dots C\}} x_{s,c}$ for every test sample $\forall s \in S$. The maximum value is used as this determines the classification. These values are separated into bins and plotted in Figure 7.3 for both the early and final exit.

The first split is between raw values associated with a correctly classified sample in green, and raw values associated with an incorrectly classified sample in red. There is significant overlap between the correct and incorrect classifications at *both* exits, with the early exit having a much more concentrated collection of low values. This can be interpreted as the networks ability (or lack thereof) to distinguish between correct and incorrect classifications. For the purposes of Early-Exit, the ideal model would have no overlap between the correct/incorrect distributions, meaning it would be possible to pick a threshold capable of perfectly determining when to exit early. Since our network has a fairly low capacity, even with more training it is unlikely to be able to better disentangle the distributions. In short, the confidence metric has an uphill struggle to determine the appropriate input samples to exit.

The same raw values are categorised according to their perceived difficulty. Figure 7.3 shows the higher difficulty samples: Difficulty 00 are samples that are misclassified at both the early and final exit and Difficulty 01 are samples misclassified at the early exit and correctly classified at the final exit. This illustrates the challenge that the confidence metric faces. Ideally, D00 samples should exit early to reduce computation without affecting accuracy, while D01 should not exit early to improve accuracy. Both distributions overlap, making it non-trivial to separate using a single-threshold confidence metric. We can see that the D01 distribution’s mass is concentrated further right at the final exit relative to the early-exit plot, but there is still no clear separation. We now use equivalent plots to explore the confidence metric values.

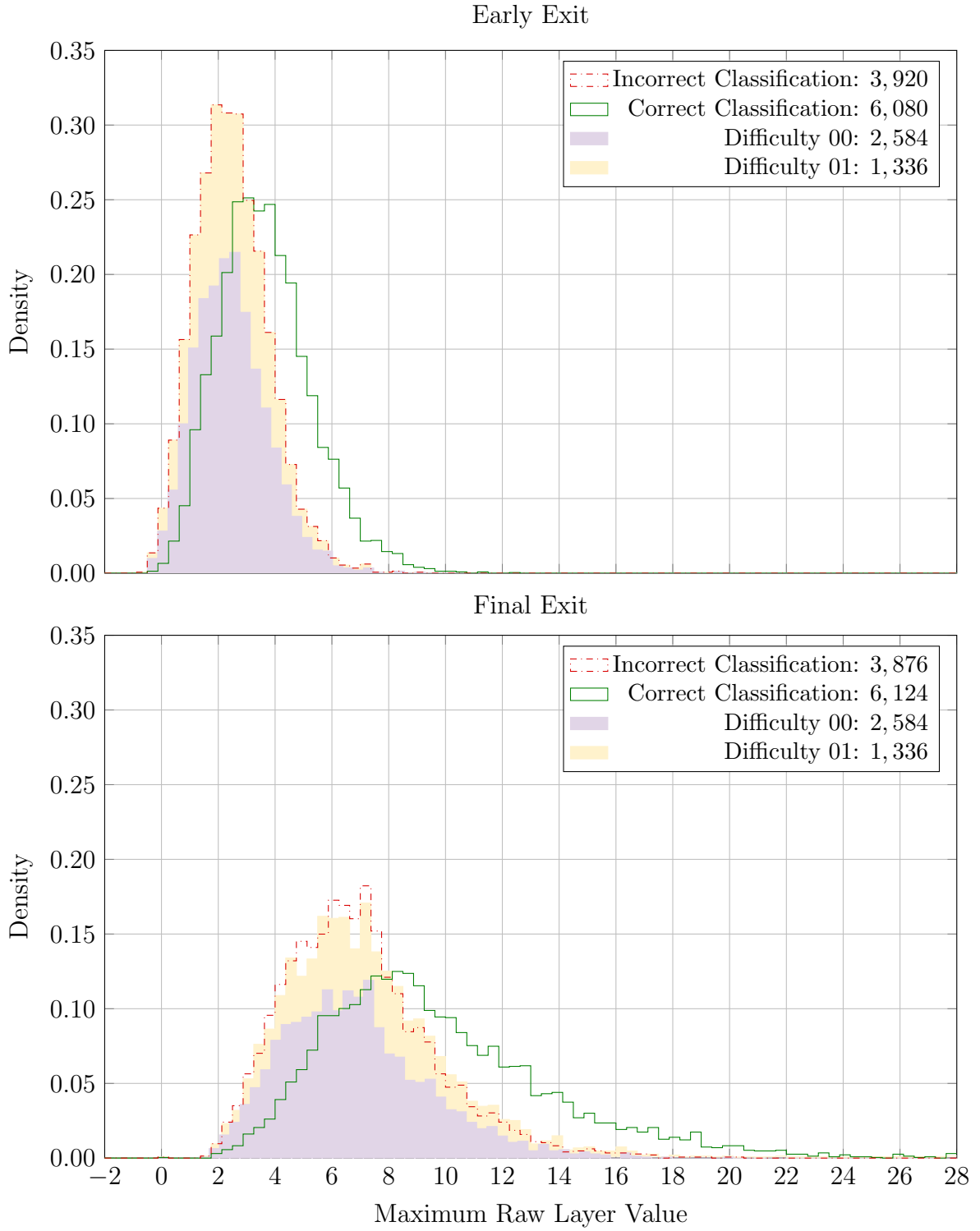


Figure 7.3: A histogram illustrating the distribution of raw values at each exit of a BranchyLeNet network trained on the CIFAR10 dataset. The red line represents the frequency of values given they are incorrectly classified at the designated exit. The green line represents the frequency of values that are correctly classified at the given exit. Difficulty 00 are the values classified incorrectly at both exits. Difficulty 01 are the values classified incorrectly at the early exit but correctly at the second exit.

7.3.2 Examining Confidence Metric Value Distributions

We introduce the five metrics under evaluation in this section:

1. *Softmax base e* is the double precision version of the original decision component. There is no truncation of input values for this metric and confidence is determined by the top1 value.
2. *Softmax base 2* is a double precision version of the original confidence metric using a base of 2 instead of e . There is no truncation of input values for this metric and confidence is determined by the top1 value.
3. *Softmax base 2, truncated* is a double precision implementation in which the non-integer part of the exponent is discarded. Confidence is determined by the top1 value.
4. *Sub max, Softmax base 2, bit-accurate* is a fixed-point precision implementation that matches the new version of the decision component. All input samples have their maxima subtracted prior to truncation through the removal of the non-integer components. The rounding and saturation behaviour of the Vivado HLS cores are replicated in software using the `fxpmath` library [104].
5. *Entropy* is a double precision implementation of the information entropy calculation applied to the results of a base e Softmax computation. While not considered for our decision component, this metric is widely used in the literature so is included for completeness.

Figure 7.4 depicts the processed value distributions for each of the confidence metrics. The x axis displays the Threshold Value for the top1-based metrics and a normalised threshold value for the entropy metric to aid comparison. For a chosen threshold on the x axis, all samples to the left would *NOT* exit early and vice versa. As with Figure 7.3, an ideal plot would perfectly separate the correct and incorrect values so that there is no distribution overlap. We would

like our chosen metric to be able to take the raw values and help separate them according to a confidence that aligns with the correct/incorrect distributions. Since perfect separation is not possible, our metric will be considered successful if it is able to exhibit a similar level of discrimination as the full precision, base e Softmax implementation.

A cursory visual inspection indicates that metrics 1. (*Softmax base e , no truncation*) and 4. (*Sub Max, Softmax base 2, bit-accurate*) seem to act as the best discriminators. Although the difference is small, 1 does this by shifting the mass of the correct samples further to the right and 4. does this by shifting the mass of the incorrect samples further to the left. 2, 3, and 4. exhibits a reduction in mass at the higher threshold values. This is due to the decrease in the value of the base for the exponential normalisation, reducing the separation between high and low raw values. 5. also provides decent separation between correct/incorrect samples but seems to have worse separation of samples with differing difficulties, than the top1-based metrics.

In general, a higher threshold value will reduce the exit rate, potentially improving accuracy and vice versa. Using the median value of each distribution we can determine the ‘best’ threshold value for a given accuracy preference. By producing the same kind of plots with the training dataset, we were able to select a threshold that separated at least 75% of the mass of the incorrect samples from the correct samples. As such, these plots are useful for determining where a threshold should be for a given dataset. Unfortunately, a simple visual inspection does not concretely reveal whether one metric should be preferred over another. We need a method, generalised away from any particular threshold value, to provide a measure of the discriminatory power of each metric. For this we turn to the AUROC and AUPRC values.

7.3.3 Applying Classifier Measures: AUROC & AUPRC

The following measures are used to determine how well a given confidence metric (coupled with the Early-Exit CNN) discriminates between input samples that should exit early and input

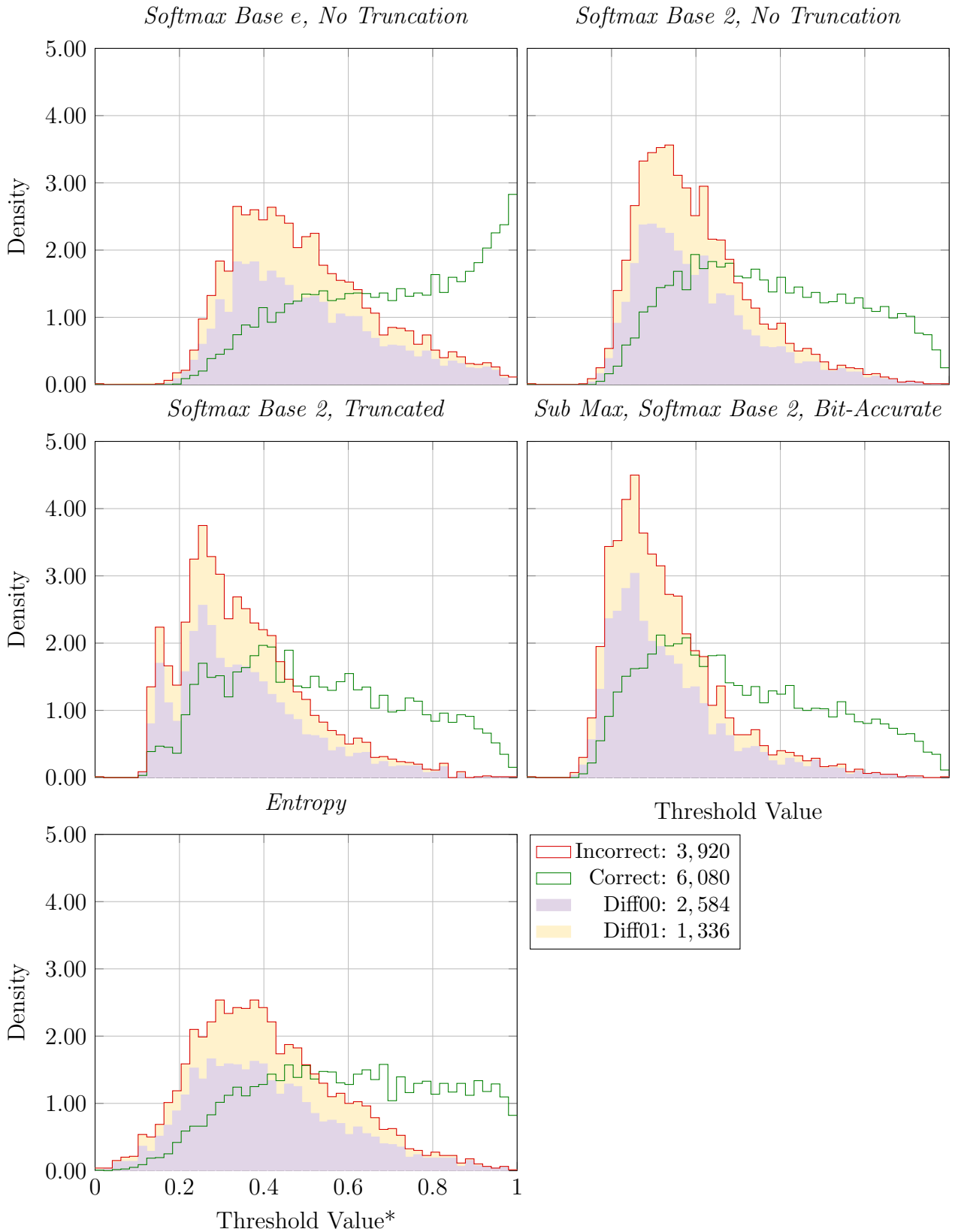


Figure 7.4: The distribution of values produced by each confidence metric computed on the early exit output of a BranchyLeNet network trained on the CIFAR10 dataset. The red line represents the frequency of values given they are incorrectly classified. The green line represents the frequency of values that are correctly classified. Diff00 are the values classified incorrectly at both exits. Diff01 are the values classified incorrectly at the early exit but correctly at the second exit. *Normalised for the entropy metric.

samples that should not. We first divide the output of every input sample into one of four categories:

- True Positive - when a sample should exit early and successfully does so. We include all samples that are correctly classified at the early exit, as well as samples that are misclassified at both exits. The latter are included in the interest of efficient computation.
- False Positive - when a sample exits early and should not have done so. This applies to samples that are incorrectly classified at the early exit and correctly classified later.
- True Negative - when a sample should not exit early and does not. This category contains samples that are incorrectly classified at the first exit and correctly classified later.
- False Negative - when a sample should not exit early and does. This includes all samples that are correctly classified at the early exit and samples that are misclassified at both exits.

AUROC

To plot the ROC curve in Figure 7.5, we iterate through a series of threshold values and determine the rate at which true positives and false positives are produced at that threshold. A truly random classifier would generate the black dash-dot line $y = x$, implying that positives are just as likely to be classified correctly as incorrectly. Fortunately, all of our confidence metrics are towards the top left, meaning that all of the confidence metrics have some ability to determine whether or not samples should exit early.

Included in the legend are the AUROC values. These values represent the area under the ROC curve, where a value of 0.5 is the baseline implying a random classifier. The key takeaway is that the difference in values is minimal. Therefore, by this measure, we are reasonably free to select the confidence metric that is most amenable to the hardware. To aid additional

analysis we include a zoomed-in plot on the part of the ROC curve with the most separation between each metric. It is interesting to see that the standard Softmax and Entropy metrics are consistently the best and worst performing. This difference implies that metrics focused primarily on measuring the separation of the top1 value from the rest, rather than metrics evaluating the distribution of values as a whole, perform slightly better for our network and task.

Metrics 1 – 3 exhibit a decline in discriminatory performance as first: the base is changed to 2, and second: the non-integer component is truncated. Metric 4. reclaims some of the performance by shifting the raw values towards the negative prior to exponential normalisation. A full precision version of metric 4. was evaluated in the same fashion but resulted in almost identical results so has been removed for brevity. This means we can more confidently say that the reclaimed accuracy is attributed to the raw value shift rather than a fluke of quantisation.

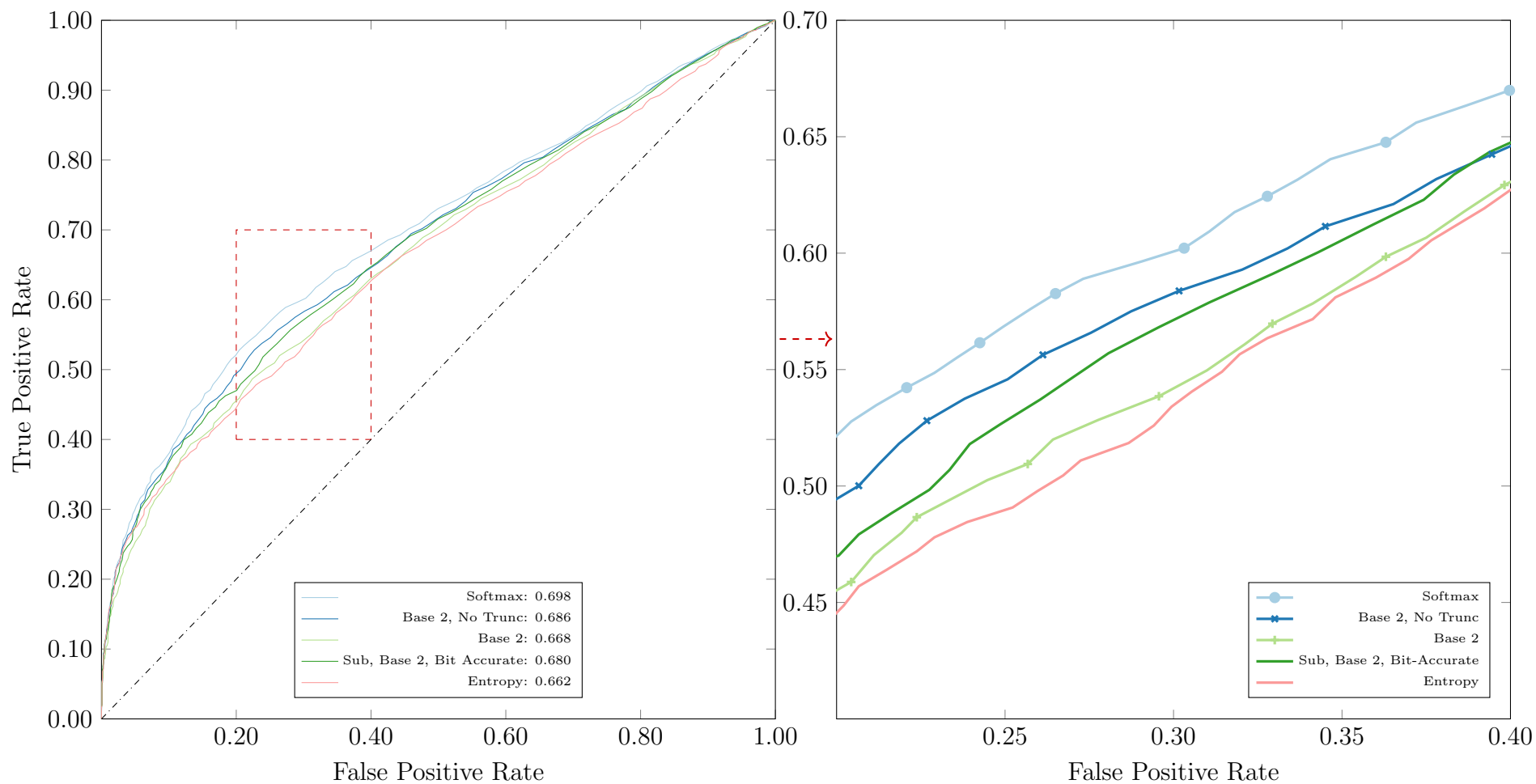


Figure 7.5: The Receiver Operating Characteristic (ROC) Curve for each confidence metric. The ROC Area Under the Curve (AUC) is a measure of a metrics ability to discriminate, shown in the legend.

AUPRC

The Recall value is equivalent to the True Positive rate determined for the ROC curves and measures the performance of the metric in making sure all early-exits exit early. Whereas, Precision is the ability of the classifier to avoid labelling samples as early-exits when they would be misclassified at the first exit but correctly classified at the final exit. Figure 7.6 depicts a similar story to the ROC curve in that the metrics all perform very similarly, again meaning we have free reign to select the most performant hardware, determined in Section 7.3.4. The nature of our experimental setup means that the baseline value for Precision is ≈ 0.86 , plotted with a dash-dot black line on the left plot of Figure 7.6. This value represents the fraction of test values that should be classified as positive. The ideal classifier will be able to correctly exit early on ‘easy’ samples without early exiting on samples that would only be correctly classified at the final exit. This would be represented by a Recall value of 1 corresponding to a Precision of 1. The AUPRC value is a measure of this performance and the fact that all metrics are similar and higher than the baseline value means that the metrics (in conjunction with the network) perform well at discriminating between ‘easy’ and ‘hard’ samples.

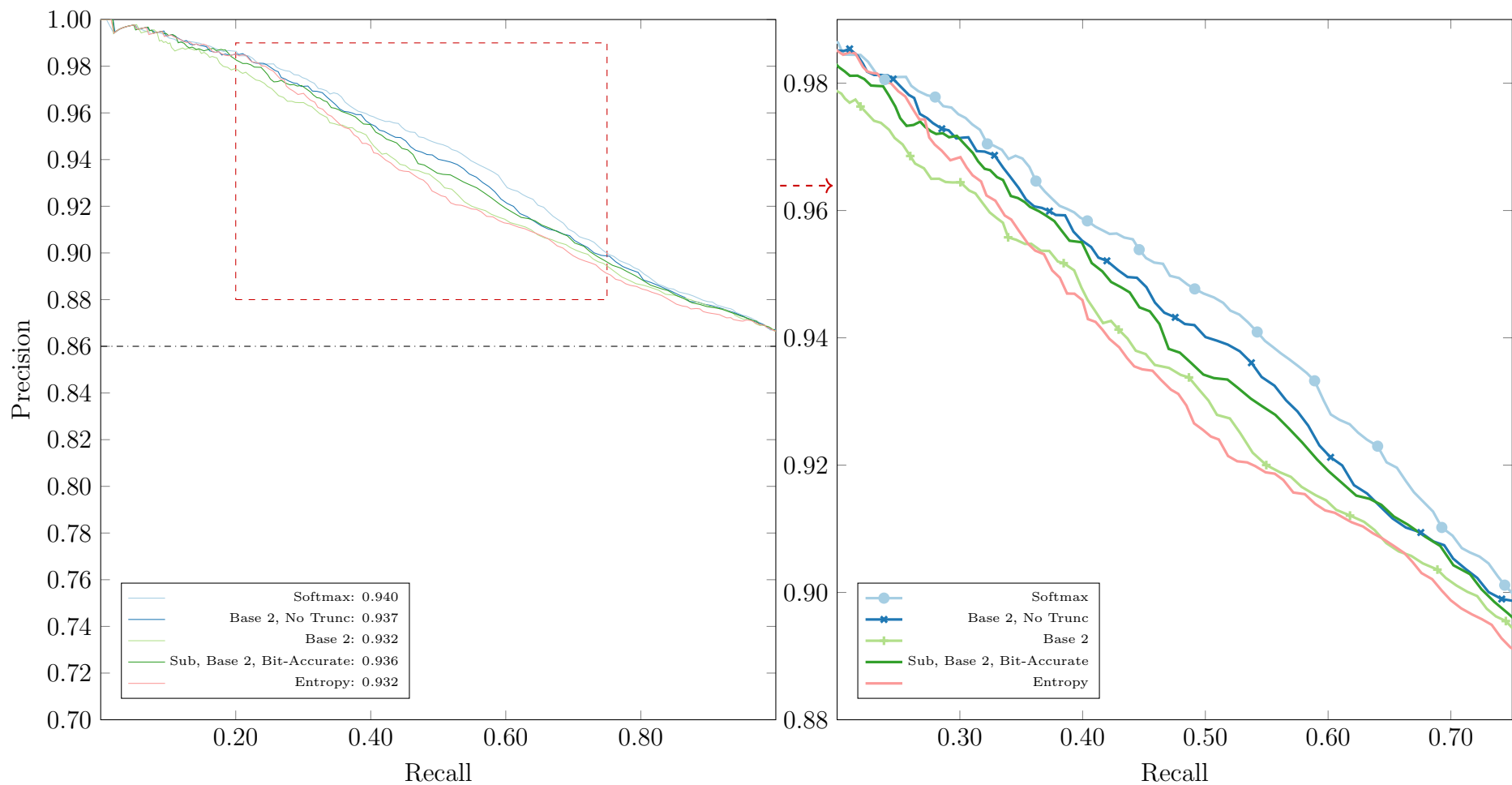


Figure 7.6: The Precision-Recall (PR) Curve for each confidence metric. The nature of our dataset means the baseline for the PR AUC is ≈ 0.86 .

Exit Rate Vs. Threshold Value

The final set of graphs, shown in Figure 7.7, plots the rate at which samples exit early for a given threshold value. As with the earlier histogram plots, the threshold value has been normalised for the entropy metric. There is more variation in the curves for this plot but this is unlikely to have an impact on the performance of the metric as a discriminator. However, from an interpretability point of view, a more linear curve makes it easier to tie a threshold value to a given exit rate. This could be utilised during the accelerator DSE optimisation to fine-tune the trade-off between accuracy and computational efficiency.

In summary, the evaluation measures show that the confidence metrics perform similarly at both full and fixed-point precision, allowing us to freely select the best hardware for the FPGA.

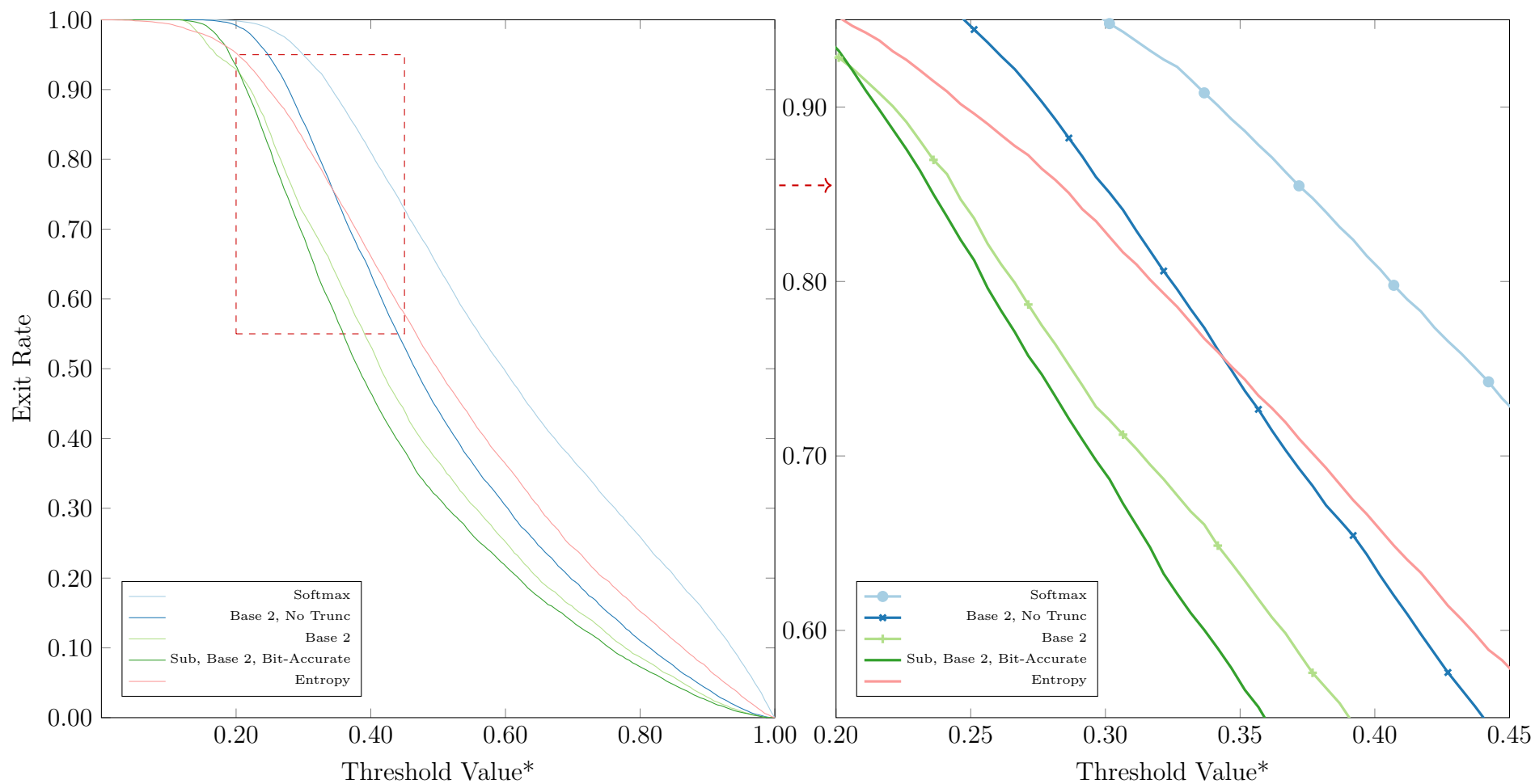


Figure 7.7: The exit rate at a given threshold value for each confidence metric.

Table 7.1: *Evaluating decision component hardware resources.*

Decision Component	Classes	Resources			
		LUTs	FFs	DSPs	BRAMs
Base e , Floating Point, Pipelined	10	5,066	4,327	12	11
No Pipeline	10	4,126	2,465	12	6
	5	3,168	1,362	0	0
Base 2, Fixed Point, Pipelined,	10	5,444	1,804	0	0
	32	4,560	1,665	0	0
	128	4,671	1,725	0	0
Subtract Max	256	4,732	2,171	0	0
	512	4,916	2,249	0	0
	1,024	4,988	2,295	0	0

7.3.4 Decision Component Hardware Statistics

By construction, the new version of the decision component is more resource efficient than the original version. Table 7.1 includes the full resource utilisation for the two original components (with and without pipelining) and the resource usage for the new decision component across a range of class values. During experimentation with the resource consumption and compilation limitations of Vivado HLS, it was found that, above 16 classes, the unrolling factor for HLS loops should be limited to 8. This provides a useful resource upper bound in the accelerator DSE, at the cost of increased individual sample latency. However, this individual sample latency is hidden through pipelining, so there is no impact on accelerator throughput for class numbers higher than 16. Looking at the resource values for the original component we note that LUT and FF usage is low, relative to the overall size of the network.

The key takeaway from Table 7.1 is that the updated decision metric not only has a reduction in LUT and FF usage, but successfully eliminates the use of DSP and BRAM blocks, as there is no need for them to perform any floating-point operations. This may be a small reduction in the total accelerator DSPs used, however, this resource is often the limiting factor in the level of unrolling/parallelism for a given network layer. For the often resource-constrained environment

of the FPGA, avoiding DSP usage benefits the rest of the accelerator considerably.

Table 7.2 details the stand-alone latency for a batch of input samples in clock cycles and the target initiation interval (II) specified at the loop level in Vivado HLS. The II is the expected number of cycles it takes for a component to be ready to receive its next input. From these values, we determine the II at the sample and pixel (individual value) level. To calculate these values, we scaled the cycle latency with respect to batch size for sample level and batch size *and* class size for pixel level. Both the original and new components that use pipelining have a much lower latency in the stand-alone case because the individual sample latency is hidden. However, in the full accelerator system the high pipeline depth of the original component actually increases the overall network latency, due to the requirement of Vivado HLS-generated kernels to be fed with enough data before producing their first output. The pipeline depth of new component depth is significantly lower, allowing the component to maintain a high throughput with a much lower latency penalty. The primary reduction in depth is a result of switching to a base 2 exponentiation operation. Using base e on a floating-point input requires $10\times$ the pipeline depth of the new Decision Component. Migrating to a fixed-point throughout the component is the second major improvement for the new Decision Component. This further decreases the pipeline depth by removing floating-point addition and comparison trees. Instead, replacing them with simple, single cycle fixed-point operations.

All these reductions in the pipeline depth of the new Decision Component directly affect the throughput of the accelerator in the following way. As samples are stored in the finite buffer and processed by the exit branch in parallel, the results for each sample need to be processed by the Decision Component before their corresponding stored result is dropped or allowed to pass through the second stage of the accelerator. The original implementation of the Decision required a minimum of 13 samples to be processed in order for the first decision to be output. This reduces the throughput of the accelerator and increases the requirements

Table 7.2: *Evaluating decision component hardware performance.*

Decision Component	Classes	Stand-alone Latency (Clock Cycles)	Initiation Interval		
			Target	Sample	Pixel
Base e , Floating Point, Pipelined	10	2,739	1	10.699	1.070
No Pipeline	10	360,359	32	351.913	35.191
	5	5,136	1	5.016	1.003
Base 2,	10	10,261	1	10.021	1.002
Fixed Point,	32	32,812	1	32.043	1.001
Pipelined,	128	131,212	1	128.137	1.001
Subtract Max	256	262,412	1	256.262	1.001
	512	524,812	1	512.512	1.001
	1,024	1,049,612	1	1024.012	1.001

for the minimum finite buffer size. By making the aforementioned changes to the Decision Component, the pipeline depth is reduced, and complexity decreased so that only a minimum of 1 - 2 samples are required to be processed before an output is produced. This improves the throughput by reducing the latency between the storing of an intermediate result and receiving the decision.

In this section, we evaluated the new decision component using a variety of measures. This gives us confidence in selecting the new component, as a replacement for the original, for accelerator designs going forward.

7.4 Summary

This chapter demonstrated the importance of hardware-aware approaches to Early-Exit CNN acceleration and provided insight into the challenge faced by confidence metrics in truly determining a DNN’s confidence in its output. We found that, even on a model that has a relatively low classification accuracy, we can still improve overall accuracy by adding early-exits. This supports the idea that DNNs are prone to ‘overthinking’ and that more efficient, input-dependent DNNs not only reduce computational cost but may improve accuracy.

Concerning the evaluated confidence metrics, the performance, in terms of discrimination between ‘easy’ and ‘hard’ samples, was similar across the board. This means that the hardware could be selected based purely on latency and resource consumption. Our results showed a small but significant reduction in resources for the new decision component as well as an improved per-sample latency, at no cost to throughput. This achieves the aim set out at the start of the chapter.

A limitation of our approach was the reliance on a single model and dataset. The results of the evaluation have the potential to change if, for example, the network had a larger capacity and was better trained, or a different dataset having an alternative distribution of ‘easy’ and ‘hard’ samples. A future study could look into these effects along with increasing the number of exits in order to provide more confidence in the confidence metrics.

CHAPTER

8

Conclusion

In Chapter 1, I highlighted the main thesis of this dissertation and the research contributions that resulted from my time spent investigating dynamic, input-dependent, Early-Exit CNNs; building upon the fpgaConvNet toolflow; creating ATHEENA to leverage probabilistic inference; and refining the toolflow with statistical modelling. I believe this work succeeded in verifying the original thesis, leading to observations and conclusions about this field of research and those adjacent to it detailed in Section 8.1. The primary aim of this dissertation was to develop a toolflow that improves the inference throughput of a pre-trained, Early-Exit CNN by automatically designing and implementing custom FPGA-based accelerators. Our specific objectives from the first chapter have been addressed across technical Chapters 3–7.

In Chapter 3 we addressed the objective of: determining a method for maximising accelerator throughput using the probabilistic nature of Early-Exit CNNs. This resulted in the contribution of the method by which resources are balanced between the different stages of an Early-Exit CNN accelerator. A method that makes use of the dynamic execution of the network across different samples, ultimately improving the throughput.

In Chapter 4 we utilised this method to address the objective of: developing an approach for mapping Early-Exit CNNs to FPGAs. This meant the realisation of a full toolflow that can receive a pre-trained Early-Exit CNN high-level description and transform it into a customised, dynamic accelerator system. To this end, we contributed custom template FPGA hardware to

execute the control and data-flow aspects of the network, a profiling tool that determines the probability of an Early-Exit, and an optimiser that uses this probability to generate a Pareto-optimal combination of design stages that produce high throughput or significant resource savings.

In Chapter 5 we addressed the toolflow’s reliance on the infinite buffer assumption. This initiated the investigation into methods used to estimate the key metrics of probabilistic systems, allowing us to tackle the objectives of: addressing the resource constraints of the FPGA device and building more rigorous probabilistic modelling into the optimiser. Coupled with the contributions from Chapter 4, the Queueing Theory-based two moment approximation of a blocking system with a finite queue, is built into the optimisation stage of ATHEENA. This resulted in a much more accurate estimation of the throughput gains given the finite resources of an FPGA device.

In the final technical chapter, 7, we address the objectives of: analysing the confidence metric to determine the impacts of reduced precision, as well as building improved template FPGA hardware components to reduce resource consumption and latency of the Decision Component. To this end, we contributed a design process motivated by adherence to the *purpose* of the Softmax function and the fundamental resource blocks of the FPGA. We simulated the proposed component, evaluating it under three measures, in order to determine the efficacy as an Early-Exit confidence discriminator. Finally, we used the knowledge gained from the simulations to produce a greatly improved Decision Component.

8.1 Observations

The diversity of DNN architectures and training methodologies (outside of the LLM/LRM space) is often driven by a requirement for task specificity, novel instances or combinations of data modes, and/or the push for real-time performance. What those solely in the Machine

Learning community can underestimate is the ramifications of increasing complexity in model training and execution. In addition to prior works, this dissertation has also shown that a shift toward dynamic, input-dependent DNN architectures is a boon for accuracy – in the reduction of ‘over-thinking’ – and for computational efficiency on standard CPU/GPUs by reducing average compute. Engineers straddling the line between the ML and (FPGA) hardware domains are uniquely placed to focus on novel hardware solutions to support and advance practical ML. There are now a wealth of interesting methodologies, like NeuraLUT [105] and Tsetlin Machines [106]–[108], that make the most of fundamental hardware resources to attain high levels of performance and efficiency. This is a strong response to the assertion that we always need *more* data and *more* power to continue improving DNN capabilities.

Hardware/Software co-design is fundamentally important to improving DNN acceleration, particularly in the FPGA-based accelerator space. Tools like fpgaConvNet [25] and ATHEENA [26] go some way towards this and have successfully broadened the deployability of Early-Exit CNNs on custom FPGA hardware. One of the key challenges in this space is the accurate estimation of resource utilisation, throughput, and latency. The variability of mapping RTL to a given device leads to significant fluctuations in DSE results, exacerbating the challenge of implementing a reliable toolflow. In short, an ML designer or hardware engineer can have great ideas about methods to improve accuracy or efficiency in isolation, but without some cross-domain cooperation, the benefits can be severely curtailed.

High-level synthesis tooling played a important role in the development of the ATHEENA toolflow. It facilitated the relatively quick implementation of custom, configurable hardware templates that formed the core functions of the accelerator designs. However, the built-in bias towards static and worst-case-scenario scheduling in many HLS tools did lead to difficulty, particularly for the conditional parts of the accelerator design. For more typical programs that better adhere to the data-flow paradigm, this bias leads to performant designs with a reduced

development cost. For the control and data-flow nature of the ATHEENA toolflow, specific control components had to be designed outside of HLS. HLS tools like Dynamatic [109] and DASS [110], which generate dynamically scheduled circuits, go a long way to making programs that rely on probabilistic execution more readily deployable. This is key in improving the templated hardware of ATHEENA. Limitations in the scale of programs able to be compiled was also a challenge with Vivado HLS, resulting in the need for compilation to be fractured manually into multiple smaller stages.

The ATHEENA toolflow has shown that there are significant throughput gains to be made by incorporating probabilistic execution information into compile-time design of hardware. However, this is only possible with some well-founded estimates on the shape of incoming data distributions and analysis grounded in statistical modelling. Queueing Theory has allowed the DSE in ATHEENA to reliably estimate the accelerator performance, improving the allocation of the key DSP and BRAM resources.

The final point I would like to stress is the benefits of open-source hardware and software. Much of my research would not have been possible without use of information and tools, like PyTorch and fpgaConvNet, that are freely available to access. I hope that my open-source code repositories, while not polished to an industry standard, might facilitate future research.

8.2 Limitations & Future Directions

There are a number of limitations with the current iteration of ATHEENA. Many of these are not immediately interesting from a research and novelty perspective and could reasonably be completed with some engineering effort. The first of these is improving the variety of supported networks and tasks. ATHEENA is somewhat limited by the support for more modern DNN architectures due to the reliance on the open-source, HLS version of fpgaConvNet. Moreover, Early-Exit networks are an increasingly small subset of the variety of input-dependent compu-

tation paradigms. Supporting alternative dynamic DNNs would require significant engineering effort to automatically translate the more complexly connected architectures in this space; however, the methodology for resource allocation in Chapter 3 could be applied to many of the coarse-grained dynamic networks with minimal adaptation.

While many aspects of the current template hardware are parameter controlled, the per-layer variable quantisation feature was not supported in ATHEENA. Though it is not the sole contributor to the issue, this does further limit the scale of networks that can be deployed using the toolflow. The issue of scale is apparent in many FPGA-based accelerator systems because of the limited resources. Toolflows like MASE [64] and Galapagos [63] avoid the issue by scattering accelerator components across multiple FPGAs. This method could be applied to ATHEENA in a similar fashion. However, I think it would present a more interesting research challenge to incorporate extreme quantisation or LUT-based DNN layers. This would help address a claim, made in Chapter 1, that Early-Exit methods and static parameter reduction methods (quantisation, pruning, and knowledge distillation) can be used together to achieve higher levels of computational efficiency whilst maintaining accuracy.

The current toolflow is limited to two exits: one early and one final exit. Existing literature points to the introduction of more intermediate classifiers (exits) enabling a more fine-grained approach to the accuracy-latency trade-off [23]. From an engineering perspective, the resource allocation and template hardware can mostly be used in their current form, with some additions to the software parsing components of the toolflow. From a research perspective, the performance analysis of throughput, with respect to allocated resources, becomes much more complicated with an increased number of exits. This is a non-trivial research direction that would likely require more advanced Queueing Theory-based statistical modelling in order to capture the variations in throughput of the accelerator as a whole.

From the machine learning side, there are works that explore hardware-aware exit place-

ment [47], learned measures to determine exit confidence [46], and hardware-constrained Neural Architecture Search for Early-Exit networks [52]. With no works of this kind targeting FPGAs (at time of writing) it would be a good research direction to pursue the inclusion of some or all of these aspects into a future version of ATHEENA. The learned confidence metric, in conjunction with NAS, would allow for a standard CNN to be fed into the toolflow augmented with Early-Exit classifiers that were both trained to achieve high levels of accuracy in line with the learned confidence metric, and optimally placed with respect to the target FPGA device. This has the potential to remove human error and bias from the placement and composition of Early-Exit classifiers. Having a learned confidence metric would benefit from incorporating statistical information from the training stage. For example, the distribution of confidence values on a per-class basis could be used to inform varying thresholds for different classes. The learned confidence metric could also be trained to reject out-of-distribution input samples that may have erroneously high confidence. In short, there are many avenues of potential future research in both the hardware and software domains relating to Early-Exit networks.

8.3 Summary

To summarise, the continued increase in the scale and ubiquity of DNNs requires new innovations in hardware to improve the deployability, efficiency, and latency – often in resource-constrained environments. With the modern networks having less and less parameter redundancy and static parameter reduction methods hitting accuracy limits, there has been an increase in focus on input-dependent DNNs. ATHEENA is an automated toolflow that leverages dynamic inference to optimally allocate the limited resources of an FPGA in order to create efficient, high throughput accelerators for Early-Exit CNNs. ATHEENA was able to achieve over $2-3\times$ the throughput of standard CNNs, often with a reduced resource consumption. This work demonstrates the value of pairing of dynamic neural networks with intelligent resource allocation for their acceleration on custom hardware devices like FPGAs.

References

- [1] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *3rd ICLR*, 2015. [Online]. Available: <http://arxiv.org/abs/1409.1556>.
- [2] C. Shan, J. Zhang, Y. Wang, and L. Xie, “Attention-based end-to-end speech recognition on voice search,” *IEEE ICASSP*, 2017. [Online]. Available: <https://api.semanticscholar.org/CorpusID:46757477>.
- [3] R. Desislavov, F. Martínez-Plumed, and J. Hernández-Orallo, “Trends in ai inference energy consumption: Beyond the performance-vs-parameter laws of deep learning,” *Sustainable Computing: Informatics and Systems*, vol. 38, p. 100 857, 2023. DOI: <https://doi.org/10.1016/j.suscom.2023.100857>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2210537923000124>.
- [4] S. Luccioni, Y. Jernite, and E. Strubell, “Power hungry processing: Watts driving the cost of ai deployment?” In *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency*, Rio de Janeiro, Brazil: Association for Computing Machinery, 2024, pp. 85–99. DOI: [10.1145/3630106.3658542](https://doi.org/10.1145/3630106.3658542). [Online]. Available: <https://doi.org/10.1145/3630106.3658542>.

- [5] M. Gupta, *Google Tensor G3: The new chip that gives your Pixel an AI upgrade*, 2023. [Online]. Available: [%5Curl%7Bhttps://blog.google/products/pixel/google-tensor-g3-pixel-8/%7D](https://blog.google/products/pixel/google-tensor-g3-pixel-8/).
- [6] J. Ortiz, M. Pupilli, S. Leutenegger, and A. J. Davison, “Bundle adjustment on a graph processor,” in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, Computer Vision Foundation / IEEE, 2020, pp. 2413–2422. DOI: [10.1109/CVPR42600.2020.00249](https://doi.org/10.1109/CVPR42600.2020.00249). [Online]. Available: https://openaccess.thecvf.com/content%5C_CVPR%5C_2020/html/Ortiz%5C_Bundle%5C_Adjustment%5C_on%5C_a%5C_Graph%5C_Processor%5C_CVPR%5C_2020%5C_paper.html.
- [7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105.
- [8] E. Wang, J. J. Davis, R. Zhao, *et al.*, “Deep neural network approximation for custom hardware: Where we’ve been, where we’re going,” *Computing Research Repository (CoRR)*, vol. abs/1901.06955, 2019. arXiv: [1901.06955](https://arxiv.org/abs/1901.06955). [Online]. Available: <http://arxiv.org/abs/1901.06955>.
- [9] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, *Pruning convolutional neural networks for resource efficient inference*, 2017. arXiv: [1611.06440](https://arxiv.org/abs/1611.06440) [cs.LG].
- [10] S. Han, J. Kang, H. Mao, *et al.*, “ESE: efficient speech recognition engine with compressed LSTM on FPGA,” *CoRR*, vol. abs/1612.00694, 2016. arXiv: [1612.00694](https://arxiv.org/abs/1612.00694). [Online]. Available: [%5Curl%7Bhttp://arxiv.org/abs/1612.00694%7D](http://arxiv.org/abs/1612.00694).

- [11] M. van Baalen, C. Louizos, M. Nagel, *et al.*, “Bayesian bits: Unifying quantization and pruning,” in *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, 2020.
- [12] R. Abra, D. Denisenko, R. Allen, T. Vanderhoek, S. Wolstencroft, and M. Gibson, “Low precision networks for efficient inference on fpgas,” in *2021 International Conference on Field-Programmable Technology (ICFPT)*, 2021, pp. 1–5. DOI: [10.1109/ICFPT52863.2021.9609837](https://doi.org/10.1109/ICFPT52863.2021.9609837).
- [13] D. Lee, S. J. Kwon, B. Kim, Y. Jeon, B. Park, and J. Yun, “Flexor: Trainable fractional quantisation,” in *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, 2020.
- [14] Y. Umuroglu, N. J. Fraser, G. Gambardella, *et al.*, “Finn,” *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Feb. 2017. DOI: [10.1145/3020078.3021744](https://doi.org/10.1145/3020078.3021744). [Online]. Available: <http://dx.doi.org/10.1145/3020078.3021744>.
- [15] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio, “Neural Networks with Few Multiplications,” *arXiv preprint arXiv:1510.03009*, 2015.
- [16] E. H. Lee, D. Miyashita, E. Chai, B. Murmann, and S. S. Wong, “LogNet: Energy-Efficient Neural Networks Using Logarithmic Computation,” in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 5900–5904. DOI: [10.1109/ICASSP.2017.7953288](https://doi.org/10.1109/ICASSP.2017.7953288).
- [17] J. Ba and R. Caruana, “Do deep nets really need to be deep?” In *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, Eds., vol. 27, Curran Associates, Inc., 2014.

- [18] G. E. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” *ArXiv*, 2015. [Online]. Available: <https://api.semanticscholar.org/CorpusID:7200347>.
- [19] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520. DOI: [10.1109/CVPR.2018.00474](https://doi.org/10.1109/CVPR.2018.00474).
- [20] M. Tan and Q. V. Le, *Efficientnet: Rethinking model scaling for convolutional neural networks*, 2020. arXiv: [1905.11946](https://arxiv.org/abs/1905.11946) [cs.LG].
- [21] J. Lin, W. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, “Mcunet: Tiny deep learning on iot devices,” in *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, 2020.
- [22] S. Teerapittayanon, B. McDanel, and H. T. Kung, “Branchynet: Fast inference via early exiting from deep neural networks,” *Computing Research Repository (CoRR)*, vol. abs/1709.01686, 2017. arXiv: [1709.01686](https://arxiv.org/abs/1709.01686). [Online]. Available: <http://arxiv.org/abs/1709.01686>.
- [23] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Weinberger, “Multi-scale dense networks for resource efficient image classification,” 2018.
- [24] S. Laskaridis, A. Kouris, and N. D. Lane, “Adaptive inference through early-exit networks: Design, challenges and directions,” in *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*, Virtual, WI, USA: Association for Computing Machinery, 2021, pp. 1–6. DOI: [10.1145/3469116.3470012](https://doi.org/10.1145/3469116.3470012). [Online]. Available: <https://doi.org/10.1145/3469116.3470012>.

- [25] S. Veneieris and C.-S. Bouganis, “Fpgaconvnet: Mapping regular and irregular convolutional neural networks on fpgas,” *IEEE Transactions on Neural Networks and Learning Systems*, Feb. 2019. DOI: [10.1109/TNNLS.2018.2844093](https://doi.org/10.1109/TNNLS.2018.2844093)..
- [26] B. Biggs, C.-S. Bouganis, and G. Constantinides, “Atheena: A toolflow for hardware early-exit network automation,” in *2023 31st International Symposium on Field Programmable Custom Computing Machines (FCCM)*, 2023.
- [27] A. Dimitriou, B. Biggs, J. Hare, and G. V. Merrett, “Fpga acceleration of dynamic neural networks: Challenges and advancements,” in *2024 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*, 2024, pp. 1–5. DOI: [10.1109/COINS61597.2024.10622127](https://doi.org/10.1109/COINS61597.2024.10622127).
- [28] M. Ren, A. Pokrovsky, B. Yang, and R. Urtasun, “Sbnet: Sparse blocks network for fast inference,” in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, Computer Vision Foundation / IEEE Computer Society, 2018, pp. 8711–8720. DOI: [10.1109/CVPR.2018.00908](https://doi.org/10.1109/CVPR.2018.00908). [Online]. Available: http://openaccess.thecvf.com/content%5C_cvpr%5C_2018/html/Ren%5C_SBNet%5C_Sparse%5C_Blocks%5C_CVPR%5C_2018%5C_paper.html.
- [29] A. Sabet, J. Hare, B. M. Al-Hashimi, and G. V. Merrett, “Temporal early exits for efficient video object detection,” *CoRR*, 2021.
- [30] B. Yang, G. Bender, Q. V. Le, and J. Ngiam, “Condconv: Conditionally parameterized convolutions for efficient inference,” in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 1305–1316. [Online]. Available: <https://proceedings.neurips.cc/paper/2019/hash/f2201f5191c4e92cc5af043eebfd0946-Abstract.html>.

- [31] H. Liu, S. Parajuli, J. Hostetler, S. Chai, and B. Bhanu, *Dynamically throttleable neural networks (tnn)*, 2020. arXiv: [2011.02836 \[cs.LG\]](https://arxiv.org/abs/2011.02836).
- [32] L. Liu and J. Deng, “Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution,” in *Association for the Advancement of Artificial Intelligence (AAAI)*, 2018.
- [33] Y. Kang, J. Hauswald, C. Gao, *et al.*, “Neurosurgeon: Collaborative intelligence between the cloud and mobile edge,” *ACM SIGARCH Computer Architecture News*, vol. 45, pp. 615–629, Apr. 2017. DOI: [10.1145/3093337.3037698](https://doi.org/10.1145/3093337.3037698).
- [34] X. Wang, F. Yu, Z. Dou, T. Darrell, and J. E. Gonzalez, “Skipnet: Learning dynamic routing in convolutional networks,” in *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIII*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., vol. 11217, Springer, 2018, pp. 420–436. DOI: [10.1007/978-3-030-01261-8_25](https://doi.org/10.1007/978-3-030-01261-8_25). [Online]. Available: https://doi.org/10.1007/978-3-030-01261-8%5C_25.
- [35] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, “Dynamic neural networks: A survey,” *IEEE transactions on pattern analysis and machine intelligence*, Oct. 2021.
- [36] Y. Wang, Y. Yue, X. Xu, *et al.*, “Adafocusv3: On unified spatial-temporal dynamic video recognition,” in *Computer Vision - ECCV 2022 - 17th European Conference, Tel Aviv, Israel, October 23-27, 2022, Proceedings, Part IV*, S. Avidan, G. J. Brostow, M. Cissé, G. M. Farinella, and T. Hassner, Eds., vol. 13664, Springer, 2022, pp. 226–243. DOI: [10.1007/978-3-031-19772-7_14](https://doi.org/10.1007/978-3-031-19772-7_14). [Online]. Available: https://doi.org/10.1007/978-3-031-19772-7%5C_14.

- [37] Y. Matsubara, M. Levorato, and F. Restuccia, *Split computing and early exiting for deep learning applications: Survey and research challenges*, 2021. arXiv: [2103.04505 \[eess.SP\]](#).
- [38] C. Lo, Y.-Y. Su, C.-Y. Lee, and S.-C. Chang, “A dynamic deep neural network design for efficient workload allocation in edge computing,” in *2017 IEEE International Conference on Computer Design (ICCD)*, 2017, pp. 273–280. DOI: [10.1109/ICCD.2017.49](#).
- [39] S. Scardapane, M. Scarpiniti, E. Baccarelli, and A. Uncini, “Why should we add early exits to neural networks?” *Cogn. Comput.*, vol. 12, no. 5, pp. 954–966, 2020. DOI: [10.1007/S12559-020-09734-4](#). [Online]. Available: <https://doi.org/10.1007/s12559-020-09734-4>.
- [40] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
- [41] B. Fang, X. Zeng, F. Zhang, H. Xu, and M. Zhang, “Flexdnn: Input-adaptive on-device deep learning for efficient mobile vision,” in *5th IEEE/ACM Symposium on Edge Computing, SEC 2020, San Jose, CA, USA, November 12-14, 2020*, IEEE, 2020, pp. 84–95. DOI: [10.1109/SEC50012.2020.00014](#). [Online]. Available: <https://doi.org/10.1109/SEC50012.2020.00014>.
- [42] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger, “Multi-scale dense convolutional networks for efficient prediction,” *Computing Research Repository (CoRR)*, vol. abs/1703.09844, 2017. arXiv: [1703.09844](#). [Online]. Available: <http://arxiv.org/abs/1703.09844>.
- [43] G. Kim and J. Park, “Low cost early exit decision unit design for cnn accelerator,” in *2020 International SoC Design Conference (ISOCC)*, 2020, pp. 127–128. DOI: [10.1109/ISOCC50952.2020.9333079](#).

- [44] Q. Zhang, J. Cao, S. Zhang, Q. Zhang, Y. Zhang, and Y. Wang, “Efficient fpga implementation of softmax layer in deep neural network,” in *2020 IEEE 5th International Conference on Signal and Image Processing (ICSIP)*, 2020, pp. 935–939. DOI: [10.1109/ICSIP49896.2020.9339290](https://doi.org/10.1109/ICSIP49896.2020.9339290).
- [45] M. Hein, M. Andriushchenko, and J. Bitterwolf, “Why relu networks yield high confidence predictions far away from the training data and how to mitigate the problem,” in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 41–50. DOI: [10.1109/CVPR.2019.00013](https://doi.org/10.1109/CVPR.2019.00013).
- [46] J. Pomponi, S. Scardapane, and A. Uncini, “A probabilistic re-interpretation of confidence scores in multi-exit models,” *Entropy*, vol. 24, no. 1, p. 1, 2021.
- [47] S. Laskaridis, S. Venieris, H. Kim, and N. Lane, “Hapi hardware-aware progressive inference,” *Proceedings of the 39th International Conference on Computer-Aided Design*, Nov. 2020. DOI: [10.1145/3400302.3415698](https://doi.org/10.1145/3400302.3415698). [Online]. Available: <http://dx.doi.org/10.1145/3400302.3415698>.
- [48] A. Kouris, S. I. Venieris, S. Laskaridis, and N. D. Lane, “Multi-exit semantic segmentation networks,” *Computing Research Repository (CoRR)*, vol. abs/2106.03527, 2021. arXiv: [2106.03527](https://arxiv.org/abs/2106.03527). [Online]. Available: <https://arxiv.org/abs/2106.03527>.
- [49] I. Leontiadis, S. Laskaridis, S. Venieris, and N. Lane, “It’s always personal: Using early exits for efficient on-device cnn personalisation,” in *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications*, Virtual, United Kingdom: Association for Computing Machinery, 2021, pp. 15–21. DOI: [10.1145/3446382.3448359](https://doi.org/10.1145/3446382.3448359). [Online]. Available: <https://doi.org/10.1145/3446382.3448359>.

- [50] H. Li, H. Zhang, X. Qi, R. Yang, and G. Huang, *Improved techniques for training adaptive deep networks*, 2019. arXiv: [1908.06294 \[cs.CV\]](#).
- [51] M. Phuong and C. Lampert, “Distillation-based training for multi-exit architectures,” in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 1355–1364. DOI: [10.1109/ICCV.2019.00144](#).
- [52] M. Gambella, J. Pomponi, S. Scardapane, and M. Roveri, “NACHOS: neural architecture search for hardware constrained early exit neural networks,” *CoRR*, vol. abs/2401.13330, 2024. DOI: [10.48550/ARXIV.2401.13330](#). arXiv: [2401.13330](#). [Online]. Available: <https://doi.org/10.48550/arXiv.2401.13330>.
- [53] I. Mitrani, *Probabilistic modelling*, eng. Cambridge: Cambridge University Press, 1998.
- [54] D. G. Kendall, “Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain,” *The Annals of Mathematical Statistics*, vol. 24, no. 3, pp. 338–354, 1953. DOI: [10.1214/aoms/1177728975](#). [Online]. Available: <https://doi.org/10.1214/aoms/1177728975>.
- [55] D. Gross, *Fundamentals of queueing theory*. eng, 4th ed. / by Donald Gross ... [et al.]. Hoboken, N.J: Wiley, 2008.
- [56] H. Styles and W. Luk, “Branch optimisation techniques for hardware compilation,” in *Field Programmable Logic and Application*, P. Y. K. Cheung and G. A. Constantinides, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 324–333.
- [57] J.-M. Garcia, O. Brun, and D. Gauchard, “Transient analytical solution of m/d/1/n queues,” *Journal of Applied Probability*, vol. 39, no. 4, pp. 853–864, 2002. DOI: [10.1239/jap/1037816024](#).

- [58] R. O. Onvural, “Survey of closed queueing networks with blocking,” *ACM Comput. Surv.*, vol. 22, no. 2, pp. 83–121, 1990. DOI: [10.1145/78919.78920](https://doi.org/10.1145/78919.78920). [Online]. Available: <https://doi.org/10.1145/78919.78920>.
- [59] J. MacGregor Smith, “Optimal design and performance modelling of m/g/1/k queueing systems,” *Mathematical and Computer Modelling*, vol. 39, no. 9, pp. 1049–1081, 2004. DOI: [https://doi.org/10.1016/S0895-7177\(04\)90534-1](https://doi.org/10.1016/S0895-7177(04)90534-1). [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0895717704905341>.
- [60] A. Boutros and V. Betz, “Fpga architecture: Principles and progression,” *IEEE Circuits and Systems Magazine*, vol. 21, no. 2, pp. 4–29, 2021. DOI: [10.1109/MCAS.2021.3071607](https://doi.org/10.1109/MCAS.2021.3071607).
- [61] M. Li, Y. Liu, X. Liu, *et al.*, “The deep learning compiler: A comprehensive survey,” *IEEE Trans. Parallel Distributed Syst.*, vol. 32, no. 3, pp. 708–727, 2021. DOI: [10.1109/TPDS.2020.3030548](https://doi.org/10.1109/TPDS.2020.3030548). [Online]. Available: <https://doi.org/10.1109/TPDS.2020.3030548>.
- [62] S. Veneieris, A. Kouris, and C.-S. Bouganis, “Toolflows for mapping convolutional neural networks on fpgas: A survey and future directions,” *ACM Computing Surveys*, Feb. 2018. DOI: [10.1145/nnnnnnnn.nnnnnnnn](https://doi.org/10.1145/nnnnnnnn.nnnnnnnn).
- [63] N. Tarafdar, N. Eskandari, V. Sharma, C. Lo, and P. Chow, “Galapagos: A full stack approach to fpga integration in the cloud,” *IEEE Micro*, vol. 38, no. 6, pp. 18–24, 2018.
- [64] C. Zhang, J. Cheng, Z. Yu, and Y. Zhao, “MASE: An efficient representation for software-defined ML hardware system exploration,” in *Machine Learning for Systems 2023*, 2023.
- [65] K. Guo, L. Sui, J. Qiu, *et al.*, “Angel-eye: A complete design flow for mapping cnn onto embedded fpga,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35–47, 2018. DOI: [10.1109/TCAD.2017.2705069](https://doi.org/10.1109/TCAD.2017.2705069).

- [66] A. Boutros, E. Nurvitadhi, and V. Betz, “Specializing for efficiency: Customizing ai inference processors on fpgas,” in *2021 International Conference on Microelectronics (ICM)*, 2021, pp. 62–65. DOI: [10.1109/ICM52667.2021.9664938](https://doi.org/10.1109/ICM52667.2021.9664938).
- [67] M. Hall and V. Betz, “From tensorflow graphs to luts and wires: Automated sparse and physically aware cnn hardware generation,” in *International Conference on Field-Programmable Technology, (IC)FPT 2020, Maui, HI, USA, December 9-11, 2020*, IEEE, 2020, pp. 56–65. DOI: [10.1109/ICFPT51103.2020.00017](https://doi.org/10.1109/ICFPT51103.2020.00017). [Online]. Available: [5Curl%7Bhttps://doi.org/10.1109/ICFPT51103.2020.00017%7D](https://doi.org/10.1109/ICFPT51103.2020.00017).
- [68] F. Qararyah, M. W. Azhar, and P. Trancoso, “An efficient hybrid deep learning accelerator for compact and heterogeneous cnns,” *ACM Trans. Archit. Code Optim.*, vol. 21, no. 2, Feb. 2024. DOI: [10.1145/3639823](https://doi.org/10.1145/3639823). [Online]. Available: <https://doi.org/10.1145/3639823>.
- [69] E. A. Lee and D. G. Messerschmitt, “Synchronous data flow,” *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987. DOI: [10.1109/PROC.1987.13876](https://doi.org/10.1109/PROC.1987.13876).
- [70] D. Navarro, Ó. Lucía, L. A. Barragán, I. Urriza, and Ó. Jiménez, “High-level synthesis for accelerating the fpga implementation of computationally demanding control algorithms for power converters,” *IEEE Transactions on Industrial Informatics*, vol. 9, no. 3, pp. 1371–1379, 2013. DOI: [10.1109/TII.2013.2239302](https://doi.org/10.1109/TII.2013.2239302).
- [71] Facebook, *Open neural network exchange*, 2017. [Online]. Available: <https://onnx.ai/>.
- [72] A. Montgomerie-Corcoran, Z. Yu, and C. Bouganis, “SAMO: optimised mapping of convolutional neural networks to streaming architectures,” *CoRR*, vol. abs/2112.00170, 2021. arXiv: [2112.00170](https://arxiv.org/abs/2112.00170). [Online]. Available: [5Curl%7Bhttps://arxiv.org/abs/2112.00170%7D](https://arxiv.org/abs/2112.00170).

- [73] Z. Yu and C.-S. Bouganis, “Streamsvd: Low-rank approximation and streaming accelerator co-design,” in *2021 International Conference on Field-Programmable Technology (ICFPT)*, 2021, pp. 1–9. DOI: [10.1109/ICFPT52863.2021.9609813](https://doi.org/10.1109/ICFPT52863.2021.9609813).
- [74] A. Montgomerie-Corcoran, S. I. Venieris, and C.-S. Bouganis, “Power-aware fpga mapping of convolutional neural networks,” in *2019 International Conference on Field-Programmable Technology (ICFPT)*, 2019, pp. 327–330. DOI: [10.1109/ICFPT47387.2019.00059](https://doi.org/10.1109/ICFPT47387.2019.00059).
- [75] G. D. Guglielmo, J. M. Duarte, P. C. Harris, *et al.*, “Compressing deep neural networks on fpgas to binary and ternary precision with HLS4ML,” *CoRR*, vol. abs/2003.06308, 2020. arXiv: [2003.06308](https://arxiv.org/abs/2003.06308). [Online]. Available: [5Curl%7Bhttps://arxiv.org/abs/2003.06308%7D](https://arxiv.org/abs/2003.06308).
- [76] Y. Umuroglu, N. J. Fraser, G. Gambardella, *et al.*, “FINN: A framework for fast, scalable binarized neural network inference,” *CoRR*, vol. abs/1612.07119, 2016. arXiv: [1612.07119](https://arxiv.org/abs/1612.07119). [Online]. Available: [5Curl%7Bhttp://arxiv.org/abs/1612.07119%7D](http://arxiv.org/abs/1612.07119).
- [77] A. Kouris, S. Veneieris, and C.-S. Bouganis, “Cascadecnn: Pushing the performance limits of quantisation in convolutional neural networks,” in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, 2018, pp. 155–1557. DOI: [10.1109/FPL.2018.00034](https://doi.org/10.1109/FPL.2018.00034).
- [78] A. Kouris, S. I. Venieris, and C.-S. Bouganis, “A throughput-latency co-optimised cascade of convolutional neural network classifiers,” in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020, pp. 1656–1661. DOI: [10.23919/DATE48585.2020.9116248](https://doi.org/10.23919/DATE48585.2020.9116248).

- [79] M. Wang, J. Mo, J. Lin, Z. Wang, and L. Du, “Dynexit: A dynamic early-exit strategy for deep residual networks,” in *2019 IEEE International Workshop on Signal Processing Systems (SiPS)*, 2019, pp. 178–183. DOI: [10.1109/SiPS47522.2019.9020551](https://doi.org/10.1109/SiPS47522.2019.9020551).
- [80] M. Farhadi, M. Ghasemi, and Y. Yang, “A novel design of adaptive and hierarchical convolutional neural networks using partial reconfiguration on fpga,” in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, 2019, pp. 1–7. DOI: [10.1109/HPEC.2019.8916237](https://doi.org/10.1109/HPEC.2019.8916237).
- [81] D. Paul, J. Singh, and J. Mathew, “Hardware-software co-design approach for deep learning inference,” in *2019 7th International Conference on Smart Computing Communications (ICSCC)*, 2019, pp. 1–5. DOI: [10.1109/ICSCC.2019.8843626](https://doi.org/10.1109/ICSCC.2019.8843626).
- [82] A. Dimitriou, M. Hu, J. Hare, and G. V. Merrett, “Exploration of decision sub-network architectures for fpga-based dynamic dnns,” in *DATE*, 2023.
- [83] G. Korol, M. G. Jordan, M. B. Rutzig, and A. C. S. Beck, “Adaflow: A framework for adaptive dataflow cnn acceleration on fpgas,” in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022, pp. 244–249. DOI: [10.23919/DATE54114.2022.9774727](https://doi.org/10.23919/DATE54114.2022.9774727).
- [84] Y. Gao, B. Zhang, X. Qi, and H. K.-H. So, “Dpacs: Hardware accelerated dynamic neural network pruning through algorithm-architecture co-design,” in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, Vancouver, BC, Canada: Association for Computing Machinery, 2023, pp. 237–251. DOI: [10.1145/3575693.3575728](https://doi.org/10.1145/3575693.3575728). [Online]. Available: <https://doi.org/10.1145/3575693.3575728>.

- [85] G. Korol, M. G. Jordan, M. B. Rutzig, J. Castrillon, and A. C. S. Beck, “Pruning and early-exit co-optimization for cnn acceleration on fpgas,” in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023.
- [86] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, “An automatic rtl compiler for high-throughput fpga implementation of diverse deep convolutional neural networks,” in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–8. DOI: [10.23919/FPL.2017.8056824](https://doi.org/10.23919/FPL.2017.8056824).
- [87] P. Contributors, *TorchScript Language Reference*, 2019. [Online]. Available: https://pytorch.org/docs/stable/jit%5C_language%5C_reference.html%5C#language-reference.
- [88] L. Deng, “The mnist database of handwritten digit images for machine learning research,” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.
- [89] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, ch. 6. [Online]. Available: <https://www.deeplearningbook.org/contents/mlp.html>.
- [90] Xilinx, *ZC706 Evaluation Board*, 2016. [Online]. Available: <https://www.xilinx.com/products/boards-and-kits/ek-z7-zc706-g.html%5C#information%7D>.
- [91] T. Hu, T. Chen, H. Wang, and Z. Wang, “Triple wins: Boosting accuracy, robustness and efficiency together by enabling input-adaptive inference,” *Computing Research Repository (CoRR)*, vol. abs/2002.10025, 2020. arXiv: [2002.10025](https://arxiv.org/abs/2002.10025). [Online]. Available: <https://arxiv.org/abs/2002.10025%7D>.
- [92] V. N. Alex Krizhevsky and G. Hinton, *Learning multiple layers of features from tiny images*, 2009. [Online]. Available: <https://www.cs.toronto.edu/~kriz/cifar.html>.

- [93] C. Banbury, V. J. Reddi, P. Torelli, *et al.*, “Mlperf tiny benchmark,” *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.
- [94] M. A. mnmostafa, *Tiny imagenet*, 2017. [Online]. Available: <https://kaggle.com/competitions/tiny-imagenet>.
- [95] M. Harchol-Balter, *Performance modeling and design of computer systems : queueing theory in action*, eng. Cambridge: Cambridge University Press, 2013.
- [96] F. Cruz, T. Van Woensel, and J. M. Smith, “Buffer and throughput trade-offs in m/g/1/k queueing networks: A bi-criteria approach,” *International Journal of Production Economics*, vol. 125, no. 2, pp. 224–234, 2010. DOI: <https://doi.org/10.1016/j.ijpe.2010.02.017>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0925527310000836>.
- [97] P. S. Foundation, *cProfile Library Reference*, 2001. [Online]. Available: <https://docs.python.org/3/library/profile.html>.
- [98] P. S. Foundation, *timeit Library Reference*, 2001. [Online]. Available: <https://docs.python.org/3/library/timeit.html>.
- [99] A. Xilinx, *AMD Xilinx FPGA Specifications*, 2024. [Online]. Available: <https://docs.amd.com/api/khub/documents/dGU6Y~1b8XPqDFk5ulti6g/content?Ft-Calling-App=ft%2Fturnkey-portal&Ft-Calling-App-Version=4.3.31#G1265242>.
- [100] R. Draelos, *Measuring performance: Auc (auROC)*, 2019. [Online]. Available: <https://glassboxmedicine.com/2019/02/23/measuring-performance-auc-auROC/>.
- [101] K. Boyd, K. H. Eng, and D. Page, “Area under the precision-recall curve: Point estimates and confidence intervals,” in *ECML/PKDD*, 2013.

- [102] R. Draelos, *Measuring performance: Auprc and average precision*, 2019. [Online]. Available: <https://glassboxmedicine.com/2019/03/02/measuring-performance-auprc/>.
- [103] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd International Conference on Machine Learning*, Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 2006, pp. 233–240. DOI: [10.1145/1143844.1143874](https://doi.org/10.1145/1143844.1143874). [Online]. Available: <https://doi.org/10.1145/1143844.1143874>.
- [104] F. A. et al., *Fxpmath: A python library for fractional fixed-point (base 2) arithmetic and binary manipulation with numpy compatibility*, 2020. [Online]. Available: <https://github.com/francof2a/fxpmath>.
- [105] M. Andronic and G. A. Constantinides, *Neuralut: Hiding neural network density in boolean synthesizable functions*, 2024. arXiv: [2403.00849](https://arxiv.org/abs/2403.00849) [cs.AR].
- [106] O.-C. Granmo, "The Tsetlin Machine - A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic," *arXiv*, 2018. [Online]. Available: <https://arxiv.org/abs/1804.01508>.
- [107] R. Shafik, A. Wheeldon, and A. Yakovlev, "Explainability and Dependability Analysis of Learning Automata based AI Hardware," in *IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, IEEE, 2020.
- [108] T. Rahman, G. Mao, S. Maheshwari, R. Shafik, and A. Yakovlev, "Matador: Automated system-on-chip tsetlin machine design generation for edge applications," in *2024 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2024, pp. 1–6. DOI: [10.23919/DATE58400.2024.10546779](https://doi.org/10.23919/DATE58400.2024.10546779).

- [109] L. Josipović, R. Ghosal, and P. Ienne, “Dynamically scheduled high-level synthesis,” in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey, CALIFORNIA, USA: Association for Computing Machinery, 2018, pp. 127–136. DOI: [10.1145/3174243.3174264](https://doi.org/10.1145/3174243.3174264). [Online]. Available: <https://doi.org/10.1145/3174243.3174264>.
- [110] J. Cheng, L. Josipović, G. A. Constantinides, P. Ienne, and J. Wickerson, “Combining dynamic & static scheduling in high-level synthesis,” in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Seaside, CA, USA: ACM, 2020, pp. 288–298. DOI: [10.1145/3373087.3375297](https://doi.org/10.1145/3373087.3375297).