THE UNIVERSITY *of* EDINBURGH

Towards Real-Time, Machine Learning enhanced Signal Detection in 5G-NR

Stefan Brennsteiner

Doctor of Philosophy

The University of Edinburgh

2023

Abstract

Mobile communication technology is essential in our connected society. In 2022, the deployment of the 5th Generation New Radio (5G-NR) standard is well on its way in many countries. 5G-NR supports new use cases and targets improvements on critical metrics such as latency, peak throughput, reliability, spectral efficiency, and more. Signal Detection - the process of recovering sent symbols from noisy observations - is one of the core tasks in the physical layer of 5G-NR.

In recent years, Machine Learning (ML) has been proposed to achieve higher detection performance on detection problems such as Orthogonal Frequency-Division Multiplexing (OFDM) and massive Multiple-Input Multiple Output (MIMO). Since it is impossible to model and design all aspects of the communication link perfectly, the idea is to allow the detection algorithms to be flexible and learn from previous observations. This can mean learning from previous detections while deployed in the field (online learning) or having been trained with example detection problems before deployment (offline training). Many proposals have been made in the literature, showing these performance gains compared to classical detectors; however, the scalability and deployability of these algorithms are typically not considered due to the recent emergence of this field.

This thesis takes steps towards closing this gap by developing real-time capable machine learning detectors for OFDM and massive MIMO. The real-time property is characterised by the timely computation of the detection task. The allowed maximum time is determined by protocol standards and system design considerations and is typically in the range of tens to a few hundred microseconds. In this work, we first identify the most promising ML detection algorithms, and their detection performance is verified in link-level simulations. Due to the high computational complexity and the low latency requirement of

the detection process, custom hardware design is required to realise real-time processing. For this reason, the algorithms are optimised for Field Programmable Gate Array (FPGA) deployment, and respective hardware architectures are proposed. The resulting digital circuits are synthesised for the Xilinx Ultrascale+ RFSoC and profiled via digital simulation.

For the OFDM detection process, a model-driven detector consisting of fully connected neural network layers is investigated. It performs joint channel estimation from pilot symbols and signal detection. The detector structure is optimised based on previous work, reducing the memory requirement by approximately four times. As on-chip memory storage and off-chip memory bandwidth are the performance limiting factors, deep compression techniques (i.e. pruning, quantisation, and Huffman coding of the weights) are investigated. Applying these techniques, the detector achieves a memory reduction of $\approx 27\times$ as compared to previous work.

Based on this, an FPGA accelerator is proposed featuring batch-processing, efficient intermediate result buffering and superscalar Huffman decompression circuits. The synthesis and simulation results show that the targeted FPGA platform can process up to 832 sub-carriers in real-time at an average detection latency of ≈ 3.7 symbols ($246.6\,\mu$s). This design shows the feasibility of processing OFDM detection in real-time; however, it also highlights current limitations, such as poor scalability to high-order modulation schemes and the still high memory requirement.

For massive MIMO, various ML-enhanced detectors using Orthogonal Approximate Message Passing (OAMP) have been proposed in the literature. These detectors are typically computationally very complex (e.g. OAMPNet) or require online training on the current channel realisation, making practical deployment difficult (e.g. MMNet). A new detector named LAMANet is proposed, which reduces computational complexity by using the simpler Approximate Message Passing (AMP) algorithm without losing performance. Further, the online training requirement is relaxed by incorporating learnable parameters in a new way and by providing precomputed initialisation matrices to the detection algorithm.

Finally, A deeply pipelined FPGA architecture is proposed for LAMANet to maximise throughput. The accelerator is synthesised, simulated, and profiled for many antenna configurations and modulation types. The pipelined latency is between $\approx 0.3\,\mu s$ for an arrangement of 32 base station antennas and eight connected users and $\approx 4\,\mu s$ for a configuration of 128 base station antennas and 64 connected users. LAMANet reaches the same throughput of $\approx 8\ \mathrm{Mbit/s}$ for a configuration of 128 base station antennas and eight connected users as a classical AMP accelerator. However, the core advantage of LAMANet is the ML enhancement which allows it to outperform classical AMP approaches vastly in terms of detection performance on realistic channels.

Lay Summary

Mobile communication is essential in our modern and connected society. Current use cases are diverse, ranging from phone calls and data streaming to industrial automation and the internet of things. A central task in the receiver of any communication system is signal detection - the task of recovering the transmitted signals in the receiver. Signal detection is challenging, as the signal might be noisy and distorted by having been transmitted over the communication channel.

Optimal solutions to the detection problem in 5G-NR are often unknown or cannot be computed in a reasonable time. For this reason, much interest has developed in using Machine Learning (ML) techniques to improve detection performance. The detection performance can be measured as the amount of correctly recovered information in the receiver. Unfortunately, many ML techniques are very complex and most previous work in the literature has not considered the timely computation of these detection algorithms when running on practical receiver hardware.

This work investigates ways to reduce complexity and compute the ML-based detectors on receiver hardware. Further, it proposes digital hardware architectures for processing these algorithms efficiently and within the timing constraints required by the 5G-NR standard documents.

For a common scenario where the cell tower transmits data to mobile users, a neural network signal detector is developed. It simplifies a previously proposed neural network by sharing layers of the network. As the memory required for storing the network weights is the limiting factor, we apply memory reduction and compression techniques. Overall, our detector requires $\approx 27\times$ less memory than previous designs. This allows us to develop a digital hardware circuit capable of computing ≈ 12.5 million detections per second on current hardware.

For another common scenario where mobile users transmit data to a cell tower equipped with many antennas, we start by providing a detailed analysis of the computational complexity of various detector proposals in the literature. Based on this, we develop a detection algorithm that incorporates ML parameters in a novel way, thereby reducing complexity. Previous proposals require the receiver to be trained while being deployed in the field. Our work found a way to mitigate this requirement, making it more suitable for practical use. Following the algorithmic investigation, a digital circuit is developed. We show that the circuit's resource usage is comparable to previous, traditional designs while providing significant detection performance improvements thanks to the ML method.

In conclusion, this work takes steps towards bringing ML enabled detectors to receiver infrastructure such as in mobile base stations. Improved detection performance benefits the various use cases of mobile communication systems and, by extension, their users.

Acknowledgements

I like to express my sincere gratitude towards my family, which always supported and encouraged me to follow my ambitions. My parents, Anna Brennsteiner and Franz Brennsteiner, know me like no one else, and I wish to become such a father someday to my future children as they have been parents to me. I sincerely thank them for all they have done and for their kind hearts. Also, I want to thank my girlfriend, Sangeetha Ratnayake, for her extensive support throughout the last four years. Experiencing challenges together made them much easier to master.

I would like to thank my supervisor Prof. Tughrul Arslan for providing guidance and advice throughout my studies. His deep knowledge of the academic environment allowed us to make the scientific contributions presented in this thesis. The Institute for Integrated Micro and Nano Systems was a great place to study, and I am grateful for the friendships formed there.

I owe many thanks to my second supervisor Prof. John Thompson whose scientific advice was invaluable on all topics related to communication science and beyond, and I admire his deep knowledge in the field. Even though he has many responsibilities, he always took the time to answer my questions patiently. I am grateful for the many things I could learn on a professional and personal level from him.

My sincere gratitude also goes to Alpha Data Parallel Systems Ltd and, in particular, to Andrew McCormick for financially supporting this project and for their advice and expertise related to Field Programmable Gate Arrays. Alpha Data has excellent products and fantastic people, and I greatly enjoyed my time there.

Finally, I would like to thank the Engineering and Physical Sciences Research Council (EPSRC) of the United Kingdom and, by extension, the British and Scottish people for funding this project. I have always felt warmly welcome in Scotland and will always remain thankful for the opportunity to live and study in such a magnificent country.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Stefan Brennsteiner

Contents

Figures and Tables

Figures

Tables

Notation

a	scalar
$\boldsymbol{a}$	vector
a_i $\boldsymbol{a}[i]$	i-th element in vector $\boldsymbol{a}$
$\boldsymbol{A}$	matrix
$\boldsymbol{A}_{i,:}$ $\boldsymbol{A}_{i,\bullet}$	i-th row of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{i,j}$	element at the i-th row and j-th column of matrix $\boldsymbol{A}$
$\boldsymbol{I_N}$	N-by-N identity matrix
$\mathcal{CN}_N(\mu, \sigma^2)$	complex-valued,N-dimensional Gaussian random variable with mean of μ and variance of σ^2
$\mathbb{C}^{M\times N}$	complex-valued matrix with M rows and N columns
$\mathbb{R}^{M\times N}$	real-valued matrix with M rows and N columns
$\mathcal{S}$	Set S
$\{\mathcal{S}\}_n$	n-th element of ordered set S
$\Re(\cdot),\ \Im(\cdot)$	a complex number's respective real and imaginary part
$\mathrm{Tr}(\boldsymbol{A})$	trace of an N-by-N square matrix, where $\mathrm{Tr}(\boldsymbol{A}) = \sum_{i=1}^{N} \boldsymbol{A}_{i,i}$
$f(\cdot)\|_x$	evaluation of function $f(\cdot)$ at x
$(\cdot)^H$	conjugate transpose of complex-valued matrix or vector
$(\cdot)^T$	real-valued transpose matrix or vector
a^*	complex conjugate of a
$E\{\cdot\}$	expectation operator
$\|\cdot\|$	absolute value
$\|\cdot\|_2$	Euclidean norm
$\|\cdot\|_2^2$	squared Euclidean norm
$\odot$	element-wise multiplication
$\oplus$	element-wise addition
$\arg\max_{\boldsymbol{x}\in\mathcal{S}} f(x)$	returns the element of set $\mathcal{S}$ which maximises $f(\cdot)$
$\arg\min_{\boldsymbol{x}\in\mathcal{S}} f(x)$	returns the element of set $\mathcal{S}$ which minimises $f(\cdot)$

Acronyms

3GPP 3rd Generation Partnership Project

5G-NR 5th Generation New Radio

6G 6th Generation

ADC Analogue to Digital Converter

ADMM Alternating Direction Method of Multipliers

AFE Analog Front End

AI Artificial Intelligence

ALU Arithmetic Logic Unit

AMP Approximate Message Passing

ANN Artificial Neural Network

ASIC Application Specific Integrated Circuit

AWGN Additive White Gaussian Noise

AXI Advanced eXtensible Interface

BA Bitwidth Activations

BER Bit Error Rate

BP Belief Propagation

BPSK Binary Phase Shift Keying

BRAM Block Random-Access Memory (RAM)

BS Base Station

BW Bitwidth Weights

CDF Cumulative Distribution Function

CLB Configurable Logic Blocks

CMOS Complementary Metal-Oxide-Semiconductor

CNN Convolutional Neural Network

CP Cyclic Prefix

CPU Central Processing Unit

CSI Channel State Information

CSI-RS Channel State Information - Reference Signal

CU Centralised Unit

DAC Digital to Analogue Converter

DC Direct Current

DCCN Deep Complex-valued Convolutional Network

DDR Double Data Rate

DFE Digital Front End

DFT Discrete Fourier Transformation

DL Deep Learning

DMRS Demodulation Reference Signal

DNN Deep Neural Network

DRAM Dynamic Random-Access Memory

DRC Design Rule Check

DSP Digital Signal Processing

DSP Digital Signal Processor

DU Distributed Unit

DUT Device Under Test

eMBB enhanced Mobile BroadBand

EP Expectation Propagation

ESN Echo State Network

FC Fully Connected

FCDNN Fully Connected Deep Neural Network

FDD Frequency Division Multiplexing

FF Flip-Flop

FFT Fast Fourier Transform

FIFO First In First Out

FPGA Field Programmable Gate Array

GAMP Generalized Approximate Message Passing

GPU Graphics Processing Unit

GSPS Giga Samples Per Second

HBM High Bandwidth Memory
HDL Hardware Description Language
HLS High Level Synthesis

ICI InterCarrier interference
IDFT Inverse Discrete Fourier Transformation
IEEE Institute of Electrical and Electronics Engineers
IFFT Inverse Fast Fourier Transform
IO-LAMA Individually Optimal LArge MIMO AMP (LAMA)
IoT Internet of Things
IP Intellectual Property
ITU-R International Telecommunication Union - Radiocommunication

LAMA LArge MIMO AMP
LLR Log-Likelihood Ratio
LMMSE Linear Minimum Mean Square Error
LNA Low Noise Amplifier
LOS Line Of Sight
LS Least Squares
LSB Least-Significant Bit
LSTM Long Short-Term Memory
LTE Long Term Evolution
LTE-A Long Term Evolution - Advanced
LUT Lookup Table

MAC Multiply-ACcumulate
MIMO Multiple-Input Multiple Output
ML Machine Learning
ML Maximum Likelihood
MLP MultiLayer Perceptron
MMSE Minimum Mean Square Error
mMTC massive Machine-Type Communication

MRC Maximum Ratio Combiner

MSB Most-Significant Bit

MSE Mean Square Error

MU-MIMO Multi User MIMO

NSE Neumann Series Expansion

OAMP Orthogonal Approximate Message Passing

OFDM Orthogonal Frequency-Division Multiplexing

OOC Out Of Context

P&R Place & Route

PA Power Amplifier

PAPR Peak-to-Average Power Ratio

PDF Probability Density Function

PGD Projected Gradient Descent

PGS Parallel Gauss-Seidel

PIPO Parallel In Parallel Out

PL Programmable Logic

PLL Phase Looked Loops

PMI Precoder-Matrix Indicator

PS Processing System

PS Penalty Sharing

QAM Quadrature Amplitude Modulation

QPSK Quadrature Phase Shift Keying

QuaDRiGa QUAsi Deterministic RadIo channel GenerAtor

RAM Random-Access Memory

RB Resource Block

RE Resource Element

ReLU Rectified Linear Unit

RF Radio Frequency

RNN Recursive Neural Network

ROM Read Only Memory

RTL Register Transfer Level

SC SubCarrier
SCS SubCarrier Spacing
SDR Software Defined Radio
SDRAM Synchronous Dynamic Random-Access Memory
SER Symbol Error Rate
SM "Streaming" Multiprocessor
sMPD simplified Message Passing Detector
SNR Signal to Noise Ratio
SoC System on Chip
SoM System on Module
SRAM Static RAM
SRS Sounding Reference Signal
STA Static Timing Analysis
SU-MIMO Single User MIMO
SVD Singular Value Decomposition

TDD Time Division Duplex
TOPS Terra Operations Per Second
TPU Tensor Processing Unit

UE User Equipment
URAM Ultra RAM
URLLC Ultra-Reliable and Low-Latency Communication

V2X Vehicle-to-everything
VLSI Very Large Scale Integration

ZF Zero Forcing

Chapter 1

Introduction

Traffic in mobile communication systems experienced fast growth in recent years. According to Ericsson's Mobility Report of June 2022, mobile network traffic has doubled in the previous two years [8]. The year-on-year growth between the first quarter of 2015 and the first quarter of 2022 has never been below 40%. Also, the number of subscribers globally is steadily rising and is forecasted to reach 9.1 billion in 2027, of which 5th Generation New Radio (5G-NR) subscriptions will amount to 4.4 billion. As the adoption of the 5G-NR technology continues, research is looking toward enabling new use cases and improving future mobile communication networks.

Many researchers envision that Machine Learning (ML) will play a significant role in implementing future mobile networks. The use of ML techniques is explored for many distinctive areas in communications systems, including network optimisation (i.e. optimising coverage, power, and capacity of the overall network), network fault detection, resource allocation, network security, and physical layer signal processing [9, 10]. The work in [11] claims that Artificial Intelligence (AI) is recognised as the most powerful enabling technology in 6th Generation (6G) mobile networks. The term AI can be understood as a superset of ML, which allows a computing system to take intelligent action. In [12], the authors envision a shift from "network softwarization" to "network intelligentiazation" and a road toward an intelligent radio. The intelligent radio ought to separate algorithms from the hardware and be capable of configuring hardware and software according to the insights gained by AI.

1.1 Motivation and Justification

Although ML technology might be beneficial in all these areas throughout the mobile communication ecosystem, the physical layer signal processing via ML technology is of particular interest as it holds the promise of revolutionising the physical layer in 6G [13, 14]. A core task in the physical layer and in any transceiver design is signal detection. Jointly with other blocks, it decisively determines the performance of the transceiver. This thesis, therefore, focuses on this signal detection and its improvements via ML methods.

1.1.1 ML-powered Signal Detection

Signal detection describes the process of recovering transmitted symbols from a noisy and potentially distorted signal in the receiver [15, chapter 8]. This process is performed in the Base Station (BS) for the uplink case and the User Equipment (UE) for the downlink case. In recent years much research showed that ML could improve detection performance over complex or unknown communication channels in either case.

Apart from introducing ML to the detection algorithm, further improvements can be achieved by removing the block boundaries of classical algorithms. An example would be to jointly perform channel estimation and signal detection in an Orthogonal Frequency-Division Multiplexing (OFDM) downlink system such as in [16]. The authors in [13] argue that the complete removal of block boundaries and replacement of the overall processing chain by an ML algorithm should be the ultimate goal for a truly disruptive physical layer technology in 6G.

Another advantage of using ML based detection algorithms is the automatic compensation of hardware impairments. The major building blocks of the transceiver such as Phase Looked Loops (PLL), Analogue to Digital Converter (ADC), Digital to Analogue Converter (DAC), Low Noise Amplifier (LNA), and Power Amplifier (PA) often exhibit certain non-ideal behaviour. They can introduce distortions and non-linearity to the received signal. ML-based detection has the potential to take these distortions into account implicitly when trained

with the appropriate data obtained from either accurate models of these components or from measurements of the hardware. For example, the work in [17] proposes a one-bit OFDM receiver with a Deep Neural Network (DNN) for channel estimation and autoencoder-based precoder and detector networks. The results show that the system can outperform simpler but unquantised designs under certain conditions.

Further, it can be argued that using well-established ML techniques and standard neural network structures can ease transceiver design. Data-driven designs mainly consist of standard machine learning blocks for which a large community of developers and many established frameworks for training and inference exist. Also, optimised hardware for fast processing is broadly available. It is easier to use these frameworks and hardware than to design and optimise specific algorithms and hardware accelerators for a specific communication use case. However, the strict real-time processing requirement combined with a short latency requirement often found in many communication systems is challenging for these algorithms, frameworks, and hardware.

1.1.2 Real-time processing

The term real-time is often used loosely in the literature. Throughout this thesis, the definition according to [18] is used: "A real-time system is one whose logical correctness is based on both the correctness of the outputs and their timeliness", or in other words, correct processing which is guaranteed to be completed within a predetermined time. This implies that (a). the processing latency has to be below this predetermined time, and (b). the throughput of the processing system has to be aligned with the data input rate to avoid the build-up of input data and, consequently, data loss and faulty behaviour.

Real-time processing for ML based detectors is challenging as the throughput requirements are high while the latency requirements are low. In the Long Term Evolution (LTE) downlink, the OFDM SubCarrier Spacing (SCS) is 15 kHz which leads to a symbol duration of $\approx 66.7\mu s$ [19]. With the new numerologies of up to 240 kHz SCS in 5G-NR, the symbol duration can be as low as $\approx 4.17\mu s$ [20]. It is not strictly required to process each symbol within one symbol time; however, latency is a major performance metric of any receiver.

Even though the research in ML-based signal detectors for application in mobile communication systems has developed rapidly in recent years, most of the proposals are based on high-level algorithmic simulations. They do not consider computational complexity or the feasibility of the detectors for real-time deployment. In particular model-driven designs have reached a level of maturity where investigation of real-time deployment and hardware implementation can be seen as the next logical step towards intelligent radio technology. Also, for data-driven design, it is of interest to understand if the current models are feasible for real-time deployment and what level optimisations can be performed. For massive Multiple-Input Multiple Output (MIMO) detection, the work in [21] presents an Application Specific Integrated Circuit (ASIC) processor based on belief propagation. A Graphics Processing Unit (GPU)-based but non-real-time autoencoder for a basic communication system is implemented in [22]. The work in [23] presents a real-time ML MIMO OFDM receiver algorithm based on reservoir computing. Also, a hardware implementation is tested with the help of two Software Defined Radios (SDR) as frontends and a connected Field Programmable Gate Array (FPGA) to handle the detector workload. Unfortunately, the hardware architecture and implementation results are not presented. Beyond these examples, real-time processing has not found much consideration in the literature yet. We identify the development of real-time capable detectors as the next logical step towards enabling ML based detectors and it is therefore at the centre of this thesis.

FPGAs will be considered throughout this thesis as the platform of choice as they provide flexibility to explore custom accelerator designs at a much lower cost than ASICs. Custom hardware design is required as latency is a major concern and cannot exceed a couple of symbol durations. GPUs, for example, have limited local memory and large memory access latency. This large latency is typically masked by multiple concurrent threads running in one execution kernel, providing good processing utilisation by fast switching between threads. Further, as this thesis explores the real-time processing feasibility of ML-based detectors, the liberty of custom logic design opens a wider design space.

1.1.3 OFDM and massive MIMO

OFDM and massive MIMO are two central technologies in 5G-NR and the respective signal detection tasks are important for the overall system performance. In LTE OFDM is used in the downlink where a transmission link is provided from the base station to the UE and detection is performed in the UE. 5G-NR added the option to also use it in the uplink. On the other hand, massive MIMO detection is typically performed in the base station. It is one of the new core technologies in 5G-NR and will most likely play an important role in future mobile communication systems.

Many proposals for ML-enhanced OFDM signal detection have been made in the literature, and they range from data-driven approaches (e.g. [16]) to enhancements of classical detectors (e.g. [24]). However, real-time deployment and its feasibility and cost are typically not considered. Similarly, many proposals for ML-enhanced massive MIMO detection have been brought forward. Typically these proposals are based on classical detectors which are then enhanced with ML parts. The underlying classical algorithms are diverse and range from Projected Gradient Descent (PGD) [25] to Orthogonal Approximate Message Passing (OAMP) [26]. Few works such as [21] consider real-time deployment and a clear need for real-time acceleration can be identified.

The fundamental motivation of this thesis is to enable ML based signal detection in future mobile communication systems. As many different algorithms for OFDM and massive MIMO detection have reached basic maturity, the logical next step towards this goal is to investigate and enable real-time processing for these algorithms.

1.2 Scope and Contribution

The scope of this thesis is the development of real-time capable signal detectors for OFDM and massive MIMO in the context of 5G-NR. The detailed scope is shown in a simplified, linear development flow of a decoder algorithm from requirements and system analysis to deployment in Fig. 1.1. Many steps and dependencies are omitted from this diagram for clarity. This thesis focuses on one of the central stages of the process, namely real-time hardware development. The work also considers parts of the neighbouring development stages, such as algorithms and hardware platforms.

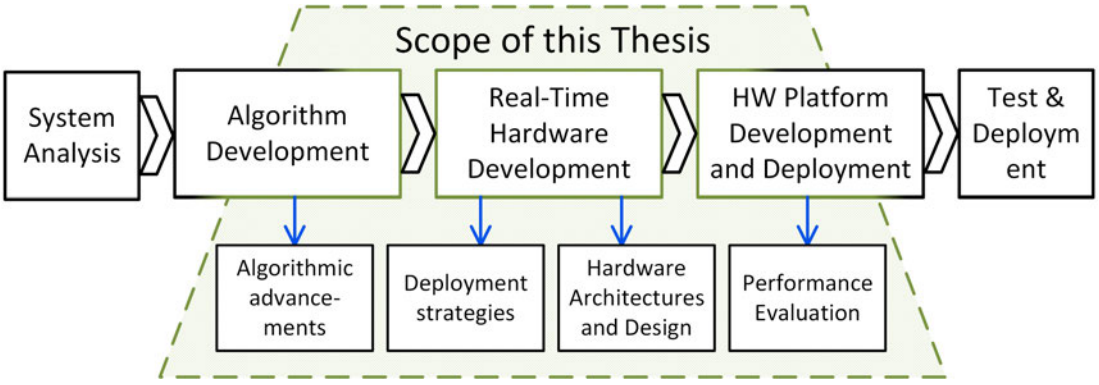


Figure 1.1: A simple development flow for detection algorithms showing the scope of this thesis and the contributions in each stage.

The contributions of the thesis can be summarised as follows and are shown at the bottom of Fig. 1.1.

Algorithmic Advancements: For the OFDM detector a layer-sharing approach is introduced to the algorithm in [16]. With this, the number of parameters in the model is reduced by ≈ 4 times which is crucial as the design is memory bound. In the uplink case, a simplified ML enhanced massive MIMO detector

is proposed based on the Approximate Message Passing (AMP) algorithm instead of the OAMP algorithm used in the literature. On top of that, the performance simulations have found that parts of the conventional AMP algorithm can be omitted when the ML enhancements are introduced - further simplifying the algorithm. Also, a novel initialisation scheme is proposed, which relaxes the online learning requirement without introducing further machine learning components to the detector. This initialisation scheme requires incorporating the learnable parameters in a new way.

Deployment Strategies: For the OFDM detector, a deep compression approach consisting of quantisation, pruning, and weight compression is evaluated to reduce the memory size further. This is the first time deep compression has been applied to a neural network in the communication domain and shows an additional memory reduction of ≈ 6.7 times. In-depth complexity analysis is provided for the uplink case, and all datatypes are carefully quantised.

Efficient Hardware Architectures: A Double Data Rate (DDR)4 memory connected custom hardware accelerator, super scalar Huffman decompressors, and a neural network processor has been implemented for the OFDM detector. The processor has multiple intermediate result buffers for optimal branching neural network workload support. A flexible hardware structure is implemented in the massive MIMO uplink case, accelerating the most common recently proposed AMP ML detectors. Another hardware design for accelerating the novel detection algorithm is implemented. It is one of the first model-based real-time detectors and shows how model-driven designs can be processed in real-time. The accelerator is implemented as a deep pipeline for maximum throughput and produces a full detection result every 108 to 1251 clock cycles, depending on the configuration. The hardware resources are comparable to conventional implementations, showing the feasibility of the ML approach.

Performance Evaluation: Both in uplink and downlink, the ML detectors deliver superior detection performance in terms of Bit Error Rate (BER) compared to traditional approaches. This is verified via link-level simulations. The developed digital circuits are verified via digital simulations before being synthesised for the target FPGA platform and profiled in terms of latency, throughput and FPGA resources.

The core contribution of this thesis is to show the feasibility of ML based signal detection in mobile communication in real-time. It provides some of the first real-time capable accelerators and evaluates their performance. It shows how important the consideration of deployability in newly proposed ML detector algorithms is.

1.2.1 Relevant Publications

During the course of this research, the following publications were produced

Covered in Chapter 3 [27]: S. Brennsteiner, T. Arslan, J. Thompson, and A. McCormick, “A Real-Time Deep Learning OFDM Receiver,” ACM Transactions on Reconfigurable Technology and Systems, vol. 15, no. 3, pp. 26:1–26:25, Dec. 2021.

Covered in Chapter 4 [28]: S. Brennsteiner, T. Arslan, J. S. Thompson, and A. McCormick, “A machine learning enhanced approximate message passing massive MIMO accelerator,” in 2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS), Jun. 2022, pp. 443–446.

Covered in Chapter 5 [29]: S. Brennsteiner, T. Arslan, J. S. Thompson, and A. McCormick, “LAMANet: A Real-Time, Machine Learning-Enhanced Approximate Message Passing Detector for Massive MIMO,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, pp. 1–14, 2022

Also, during this research, the following publication was produced; however, thematically, it is too far removed to be included in this thesis in more detail.

Not covered in this thesis [30]: S. Brennsteiner, T. Arslan, and J. Thompson, "Evaluation of Partially Constant, Fine-Grained, Dynamic Partial Reconfigurable Functions in FPGAs," in 2019 International Conference on Field-Programmable Technology (ICFPT), Dec. 2019, pp. 347–350

1.3 Thesis Outline

Chapter 2 provides background information on the most important aspects of the thesis in preparation for the following chapters. First, 5G-NR fundamentals are reviewed before introducing the two core detector technologies of 5G-NR and of this work, namely OFDM and massive MIMO. Then, machine learning basics are discussed, and a background in FPGAs is provided.

Chapter 3 introduces a novel neural network based, OFDM joint channel estimator and signal detector. The neural network structure was simplified compared to previous proposals in the literature by sharing neural network layers. Further, deep compression techniques such as low bit width quantisation, pruning, and Huffman weight compression are investigated and profiled. This significantly reduces the memory consumption and allows the design of a real-time capable FPGA accelerator. The accelerator features superscalar Huffman decoders, intermediate result buffers capable of supporting the targeted branching neural network, and batch processing capability. Simulations on the targeted Xilinx RFSoC platform show real-time capable processing of up to 832 SubCarrier (SC).

Chapter 4 reviews previously proposed, ML enhanced, massive MIMO signal detectors. First, a comprehensive review of state-of-the-art detectors is provided, including various design approaches such as data-driven designs, PGD-based designs, and OAMP-based designs. After identifying OAMP as one of the most suitable base algorithms, a unified algorithm is proposed to

implement various previously proposed detectors. Depending on the external configuration, it can implement various forms of the OAMPNet [31] and MMNet [26] detectors. Based on this, an FPGA accelerator is proposed and profiled for many antenna configurations. The pipelined latency of this accelerator is as low as ≈ 2 µs for a configuration of 32 base station antennas and eight users. Compared to previous accelerators without ML enhancements, the throughput is considerably lower, which motivates the work in the next chapter.

Chapter 5 proposes a novel AMP-based, ML-enhanced massive MIMO detector named LAMANet. Link-level simulations show that the less complex AMP algorithm can be used instead of OAMP as the base algorithm without losing significant performance. Further, one of the main problems of the previously proposed MMNet detector is its online learning requirement. This is mitigated in LAMANet by incorporating the learnable matrices in a new way. Extensive complexity analysis is provided, and a LAMANet FPGA accelerator is proposed. The accelerator is comparable to previous detectors in terms of throughput, latency, and resource usage while supporting ML enhancements, thereby delivering higher detection performance.

In **Chapter 6** conclusions and future research opportunities are discussed. Specific to the investigated detectors of OFDM and massive MIMO, future work is briefly discussed. In particular, steps to realise an over-the-air demonstrator are pointed out. Since the field of ML enhanced physical layer processing in communication systems is just emerging, future research opportunities beyond signal detection are plentiful and briefly discussed too.

Chapter 2

Background

This chapter provides the technical background in preparation for the rest of the thesis. Initially, the context of 5G-NR is briefly introduced before discussing system models and signal detection for OFDM and massive MIMO. Next, ML fundamentals are briefly introduced to provide the basis for the ML detection algorithms of the later chapters. Since real-time processing is a central part of the thesis, the need for hardware acceleration is elaborated next, where FPGAs are identified as the most suitable processing platform for this work.

2.1 5G-NR and beyond

The development of 5G-NR has been driven by three distinct use cases. These namely are enhanced Mobile BroadBand (eMBB), Ultra-Reliable and Low-Latency Communication (URLLC), and massive Machine-Type Communication (mMTC) [32, 33, 34]. The aim of eMBB is to enable applications such as streaming multimedia content, virtual reality, and more by providing high data rates. The International Telecommunication Union - Radiocommunication (ITU-R) distinguishes between the hotspot and wide area coverage case. Very high data rates are required in the former, and user mobility can be assumed to be low. An example of such a scenario is dense urban deployment. The data rate requirement might be lower in the wide area coverage case, but medium to high user mobility has to be supported. As the name suggests, the URLLC use-case has strict requirements on availability and latency. Example applications are Vehicle-to-everything (V2X) for autonomous driving, industrial automation,

real-time remote machine control, and more. Thirdly, mMTC describes a use case in which a large number of small, low-cost, and low-power devices exchange non-timing critical messages. Examples are Internet of Things (IoT) devices such as those used for home automation, consumer goods, etc.

Although 5G-NR introduces many changes in various domains from 5G-NR core network to the physical layer, commonly, the following three technologies are regarded as the main innovation to realise the ITU-R performance targets [34]:

- Massive MIMO for increased spectral efficiency and higher data rates,
- Network densification to support high data rates via local hotspots and small cells,
- New frequency bands in the mm-wave range for high throughput.

Similar to Long Term Evolution - Advanced (LTE-A), OFDM was chosen as the waveform for downlink in 5G-NR. For the uplink, OFDM with or without Fast Fourier Transform (FFT)-precoding is available [32]. However, a flexible numerology is introduced, which allows choosing one of seven SCSs between $15\,\mathrm{kHz}$ and $960\,\mathrm{kHz}$ [35].

This thesis focuses on the signal detection problem in massive MIMO and OFDM. Both technologies are integral parts of the 5G-NR physical layer and will likely also play an essential role in future mobile communication systems. Many machine learning algorithms have been recently proposed for these problems, and some researchers expect ML to become an important technique in future networks [12, 13, 36].

2.2 Orthogonal Frequency Division Multiplexing

OFDM is a ubiquitous multi-carrier transmission scheme used in many communications systems ranging from IEEE 802.11 Wi-Fi to mobile communications in LTE and 5G-NR. An introduction to OFDM can be found in most textbooks (e.g. in [37, Chapter 12]). In LTE it is used in the downlink from the base station

to the UEs. In 5G-NR OFDM can be used for uplink and downlink. One of the main advantages of OFDM is its spectral efficiency due to the orthogonality of its SC and its resilience to multipath delay spread due to the introduction of long symbol times on multiple SC. Further, the modulation of the SC can be performed very efficiently in the digital domain via the Inverse Fast Fourier Transform (IFFT) transform as shown Section 2.2.1. The main disadvantage of OFDM is its high Peak-to-Average Power Ratio (PAPR), which makes power amplifier design difficult and introduces power inefficiencies. This is why in 5G-NR, other uplink modulation schemes are still allowed. Other techniques, such as signal shaping, can reduce the PAPR as well.

The resource grid as formed by OFDM in LTE and 5G-NR is shown in Fig. 2.1. The Resource Element (RE) is the fundamental time-frequency resource in the system consisting of one symbol at one SC. Multiple RE form a Resource Block (RB), which is typically the granularity in which resources are assigned to users. The SCS is an important parameter in the numerology of an OFDM system and indicates the signal bandwidth each signal modulated on a SC can occupy.

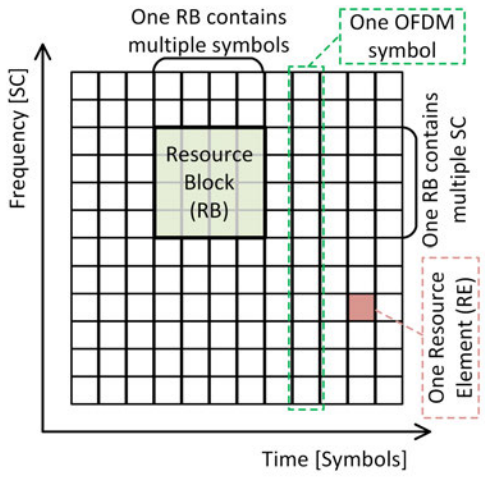
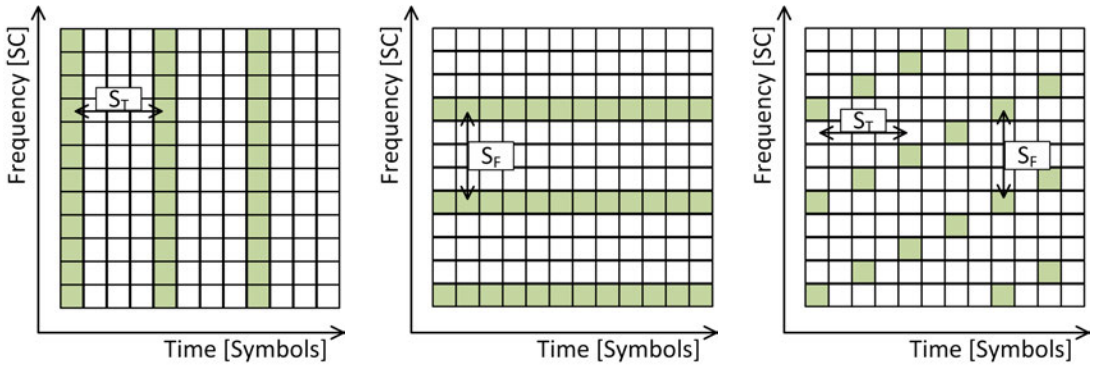


Figure 2.1: Resource Grid in LTE and 5G-NR systems.

2.2.1 System Model

This section introduces a simplified OFDM system model and briefly explains its main building blocks. The digitally modulated signal $\boldsymbol{x} \in {\mathcal{C}_\mathcal{C}}^{M_{SC}}$ is provided as input to the OFDM baseband processor in the transmitter, where M_{SC} is the number of data-carrying SCs in the OFDM system and $\mathcal{C}_\mathcal{C}$ is the complex-valued symbol constellation. The modulation ranges from Binary Phase Shift Keying (BPSK) to high-order Quadrature Amplitude Modulation (QAM) constellation. Theoretically, they can be chosen individually for each SC; however, in practice, a more coarse approach is used.

The input signal $\boldsymbol{x}$ consists of both data and pilot symbols. The pilot symbols are used for channel estimation, which aids the detection process. In mobile communications, the usage of pilot symbols for channel estimation is well established. The pilot symbols can be assigned in various ways to the resource grid as shown in Fig. 2.2.



(a) Block Type **(b)** Comb Type **(c)** Lattice Type

Figure 2.2: OFDM Pilot assignment in the resource grid. Concept from [1].

A comb-type pilot assignment as shown in Fig. 2.2b is considered. It is defined such that

$$\boldsymbol{x}[\{\mathcal{P}\}_n] := \boldsymbol{Pilot}[n];\ n \in \{1, ..., M_p\}, \tag{2.1}$$

$$\boldsymbol{x}[\{\mathcal{D}\}_n] := \boldsymbol{Data}[n];\ n \in \{1, ..., M_d\}. \tag{2.2}$$

Where M_p is the number of pilot symbols in the $\boldsymbol{Pilot}$ vector and $\mathcal{P} = \{1, P_f, 2P_f, M_pP_f\}$ are the equidistant indices of the pilots spaced on the frequency axis by P_f. In a similar way, the data indices are given by $\mathcal{D} = \{1, ..., M_{SC}\} \setminus \mathcal{P}$. In LTE a lattice-type arrangement is used [19, chapter 8].

OFDM processing is performed in the digital baseband processor and can be divided into frequency- and time-domain processing as shown in Fig. 2.3. The first processing step on the input signal $\boldsymbol{x}$ is to add guard band SCs to form the signal $\boldsymbol{x_{GB}}$. To the signal $\boldsymbol{x}$, M_{GB} zeros are prepended and appended. The guard bands are important as they prevent interference with neighbouring channels.

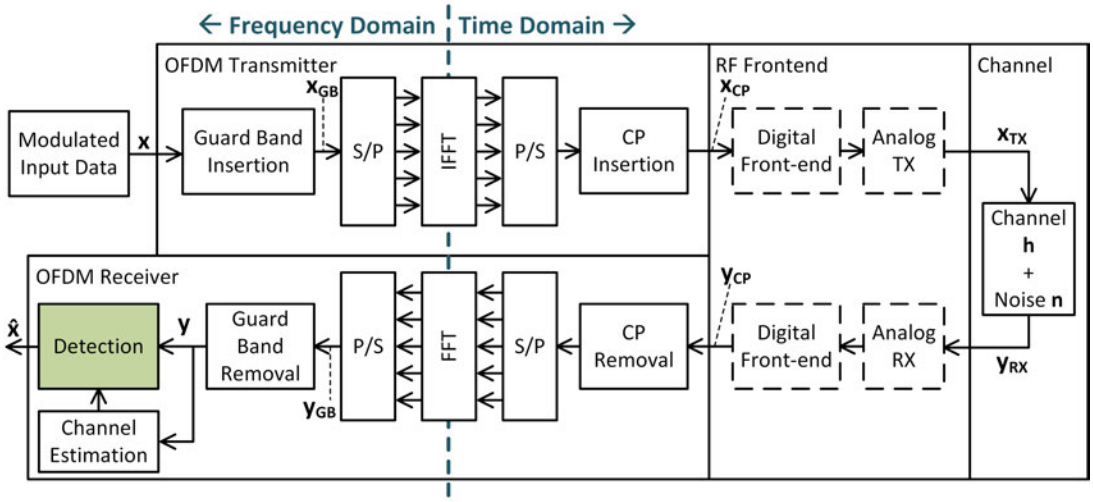


Figure 2.3: Signal processing chain in an OFDM system including the Radio Frequency (RF) frontend, the channel, and the channel estimation and signal detection.

Next, the signal $\boldsymbol{x_{GB}}$ is transformed from a serial to a parallel representation so that it can be provided as the input to the IFFT-block. In the IFFT, the transmit vector is mapped to M_{SC} equally spaced sub-carriers. In addition, the Cyclic Prefix (CP) is added

$$\boldsymbol{x_{CP}}[k] = \frac{1}{\sqrt{M_{SC} + M_{GB}}} \sum_{n=0}^{N-1} \boldsymbol{x_{GB}}[n] e^{j \frac{2\pi n(k - M_{CP})}{M_{SC} + M_{GB}}}, \tag{2.3}$$

where $k \in \{0, ..., M_{SC} + M_{GB} + M_{CP} - 1\}$ and M_{CP} is the cyclic prefix duration in samples [38]. After passing through the digital and the analogue front-end, which might perform more transformations on the signal, it passes through the channel. In the channel, the transmit signal $\mathbf{x_{TX}}$ is convolved with the channel impulse response $\boldsymbol{h}$ and the noise $\boldsymbol{n}$ is added, forming the receive signal

$$\boldsymbol{y_{RX}}[k] = \boldsymbol{x_{TX}}[k] \circledast \boldsymbol{h}[k] + \boldsymbol{n}[k]. \tag{2.4}$$

On the receiver side, Eq. (2.3) is reversed by removing the cyclic prefix and then performing an FFT to recover the complex frequency domain symbols $\boldsymbol{y_{GB}}$. Following that, the guard bands are removed, and the channel estimation and signal detection processes can start based on the signal $\boldsymbol{y}$.

2.2.2 Conventional Channel estimation

As pilot symbols are used for channel estimation, the estimation task consists of two parts. Namely, a.) estimation of the channel coefficient at the pilot carrying SC and b.) interpolation of the estimated channel coefficients to the data carrying SC.

For task a.), the Least Squares (LS) estimator and Minimum Mean Square Error (MMSE) estimator are most commonly used. In the frequency domain, an OFDM system containing only pilot symbols can be expressed as [39]

$$\begin{bmatrix} y[1] \\ y[P_f] \\ . \\ y[P_f M_p] \end{bmatrix} = \begin{bmatrix} X[1] & 0 & 0 \\ 0 & X[P_f] & 0 \\ . & . & . \\ 0 & 0 & X[P_f M_p] \end{bmatrix} \begin{bmatrix} g[1] \\ g[P_f] \\ . \\ g[P_f M_p] \end{bmatrix} + \begin{bmatrix} n[1] \\ n[P_f] \\ . \\ n[P_f M_p] \end{bmatrix}. \tag{2.5}$$

The transmitted and known pilot symbols are stored on the diagonal of $\boldsymbol{X}$, the received signal is $\boldsymbol{y}$, the frequency domain channel response is $\boldsymbol{g}$, and the noise is $\boldsymbol{n}$. In matrix notation, this is

$$\boldsymbol{y}[k] = \boldsymbol{X}[k]\boldsymbol{g}[k] + \boldsymbol{n}[k], \tag{2.6}$$

where $k \in \{1, ..., M_P\}$ are the pilot carrier indices and $\boldsymbol{g}$ is the frequency response of the channel, i.e. the M_{SC}-point FFT of $\boldsymbol{h}$. The number of channel taps in $\boldsymbol{h}$ is given by M_L. For this analysis, we let the mean of the frequency domain channel coefficients be zero $E\{g\} = 0$ and the variance $E\{|g|^2\} = M_L\sigma_g^2$. We assume $\boldsymbol{n}$ to be Additive White Gaussian Noise (AWGN) with zero mean and variance of $E\{|n|^2\} = M_P\sigma_n^2$. The channel coefficients and the noise are independent of each other.

The LS estimator, as formulated for the estimation problem in Eq. (B.8), can be rewritten for the above system equation Eq. (2.6) as

$$\begin{aligned} \widehat{\boldsymbol{g}}_{\boldsymbol{LS}} &= \arg\min_{\boldsymbol{g}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{g}\|_2^2, \\ &= (\boldsymbol{X}^H\boldsymbol{X})^{-1}\boldsymbol{X}\boldsymbol{y}, \\ &= \boldsymbol{X}^{-1}(\boldsymbol{X}^H)^{-1}\boldsymbol{X}^H\boldsymbol{y}, \\ &= \boldsymbol{X}^{-1}\boldsymbol{y}. \end{aligned} \tag{2.7}$$

Since the matrix $\boldsymbol{X}$ is diagonal, the inverse can be easily obtained by forming the reciprocal of the diagonal entries. Similarly, the MMSE estimate of the channel can be obtained following the derivation in [39] and according to Eq. (B.12)

$$\begin{aligned} \widehat{\boldsymbol{g}}_{MMSE} &= \mathbb{E}\left\{\|\boldsymbol{g} - \widehat{\boldsymbol{g}}\|_2^2\right\}\boldsymbol{y} \\ &= \mathbb{E}\left\{\boldsymbol{g}\boldsymbol{y}^H\right\}\mathbb{E}\left\{(\boldsymbol{y}\boldsymbol{y}^H)\right\}^{-1}\boldsymbol{y}, \\ &= M_L\sigma_g^2\boldsymbol{X}^H(M_L\sigma_g^2|\boldsymbol{X}|^2 + M_P\sigma_n^2)^{-1}\boldsymbol{y}, \\ &= \boldsymbol{R}_{\boldsymbol{gg}}[\boldsymbol{R}_{\boldsymbol{gg}} + (\boldsymbol{X}\boldsymbol{X}^H)^{-1}M_P\sigma_n^2]^{-1}\widehat{\boldsymbol{g}}_{\boldsymbol{LS}}, \end{aligned} \tag{2.8}$$

where the channel covariance is indicated by $\boldsymbol{R}_{\boldsymbol{gg}}$.

The second task for channel estimation in the frequency domain is to interpolate between the estimated channel values at the pilot positions. This process will obtain channel estimates at the data carriers. Depending on the pilot arrangement, different interpolation strategies might be used. In the simplest case, a block-type or comb-type pilot arrangement only requires interpolating across one dimension of the resource grid. In the lattice-type arrangement, 2D interpolation is required. One of the simplest interpolation techniques for comb-type pilot arrangements is linear interpolation, in which data carriers are estimated by

$$\widehat{\boldsymbol{g}}[\{\mathcal{P}\}_n + k] = \widehat{\boldsymbol{g}}[\{\mathcal{P}\}_n] + \frac{\widehat{\boldsymbol{g}}[\{\mathcal{P}\}_{n+1}] - \widehat{\boldsymbol{g}}[\{\mathcal{P}\}_n]}{(k/P_f)}, \quad 0 < k < P_f, \quad n \in \{1, ..., M_p\}. \tag{2.9}$$

Another, more accurate interpolation technique is second-order interpolation which interpolates by taking three neighbouring pilot locations into account [40]. Also, low-pass filtering approaches are common and easy to implement.

2.2.3 Conventional Signal Detection

The signal detection task in the frequency domain in OFDM (and in other systems) is often also referred to as equalisation. A simple one-tap detector will be sufficient if the channel is only slowly changing (i.e. the coherence interval is larger than the symbol time $T_C \geq T_S$). The Zero Forcing (ZF) detector can be formulated as [41, chapter 6.5]

$$\boldsymbol{x}[k] = \boldsymbol{g}^{-1}[k]\boldsymbol{y}[k], \quad k \in \mathcal{D}. \tag{2.10}$$

As shown in Appendix B.1.3, the ZF detector does not take noise into account, and the performance might suffer under noise enhancement in low Signal to Noise Ratio (SNR) scenarios. The MMSE detector is described as [41, chapter 6.5]

$$\boldsymbol{x}[k] = \boldsymbol{g}^*[k] \left(|\boldsymbol{g}[k]|^2 + \frac{1}{SNR} \right)^{-1} \boldsymbol{y}[k], \quad k \in \mathcal{D}. \tag{2.11}$$

In very fast fading channels ($T_C \leq T_S$), the channel changes significantly over the OFDM symbol duration. This results in InterCarrier interference (ICI) and using the simple one-tap equaliser results in performance degradation. Multi-tap equalisers can be constructed to cancel this interference. However, this scenario is not further considered in this thesis as it is unlikely to occur in 5G-NR systems.

2.3 Massive Multiple-Input Multiple Output (MIMO)

As mentioned above, massive MIMO is one of the most significant technological advancements in 5G-NR. The term "massive" thereby refers to the number of base station antennas. It is not clearly defined when a MIMO system can be categorised as massive MIMO, but base stations with 64 or more antennas are typically considered as such. Equipping the base station with this large number of antennas allows to realise one or more of the following benefits:

Beamforming describes the process in which the base station generates dedicated beams as a radiation pattern. Multiple beams can be generated and steered in space to target data transmission to the connected users. This technique allows realising a **beamforming gain** which improves the SNR as compared to omnidirectional antennas. In a similar way, as beams can be steered to users, the absence of radiation can be steered in order to avoid interference in a concept called **null steering**. A typical use case is to reduce inter-cell interference.

Spatial Multiplexing allows for higher data rates and higher spectral efficiency as each beam can carry distinct data at the same time-frequency resource. A downlink or uplink scenario with many single- or multi-antenna UEs can be an example.

While the target of spatial multiplexing is high throughput, the orthogonal target of **transmit diversity** is to implement reliable communication. Coding can be performed across the spatial dimension complementing any other coding applied.

2.3.1 System Model for massive MIMO detection

A generic massive MIMO system can be described as [31]

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{n}. \tag{2.12}$$

With the complex channel matrix $\boldsymbol{H} \in \mathbb{C}^{\overline{M_R} \times \overline{M_T}}$ where $\overline{M_R}$ is the number of antennas at the base station, and $\overline{M_T}$ is the number of single-antenna UEs. The base station receives the signal vector $y \in \mathbb{C}^{\overline{M_R}}$. The $\overline{M_T}$ single antenna UEs transmit at the same frequency to form the transmit vector $x \in \mathbb{C}^{\overline{M_T}}$. The noise vector $\overline{\boldsymbol{n}}$ consists of $\overline{M_R}$ i.i.d. complex Gaussian entries such that $\boldsymbol{n} \sim \mathcal{CN}_{\overline{M_R}}(0, \sigma^2)$. We assume the condition for massive MIMO holds, where $\overline{M_T} << \overline{M_R}$ and where $\overline{M_R}$ is sufficiently large. The symbol values each entry of $\overline{\boldsymbol{x}}$ can take are defined by the set $\mathcal{C}_\mathcal{C} \subset \mathbb{C}$ with $|\mathcal{C}_\mathcal{C}| = M_C \times M_C$. The parameter M_C describes the number of constellation points of a given modulation type on one axis of the complex plane, such that for symmetrical schemes $\mathcal{C}_\mathcal{R} = \Re(\mathcal{C}_\mathcal{C}) = \Im(\mathcal{C}_\mathcal{C})$ and $|\mathcal{C}_\mathcal{R}| = M_C$. The number of bits contained in a symbol is $N_{bits} = \log_2(M_C \times M_C)$. For example, the QAM64 modulation has $M_C = \mathrm{sqrt}(64) = 8$ unique numerical values per axis in the complex plain and can transport $N_{bits} = \log_2(64) = 6$ bits per symbol.

The operating principle of a massive MIMO system is illustrated in Fig. 2.4. Each illustrated connection between each BS and UE antenna is represented by a complex-valued channel coefficient in the channel matrix $\boldsymbol{H}$. Each column of $\boldsymbol{H}$ holds all amplitude and phase values for one UE.

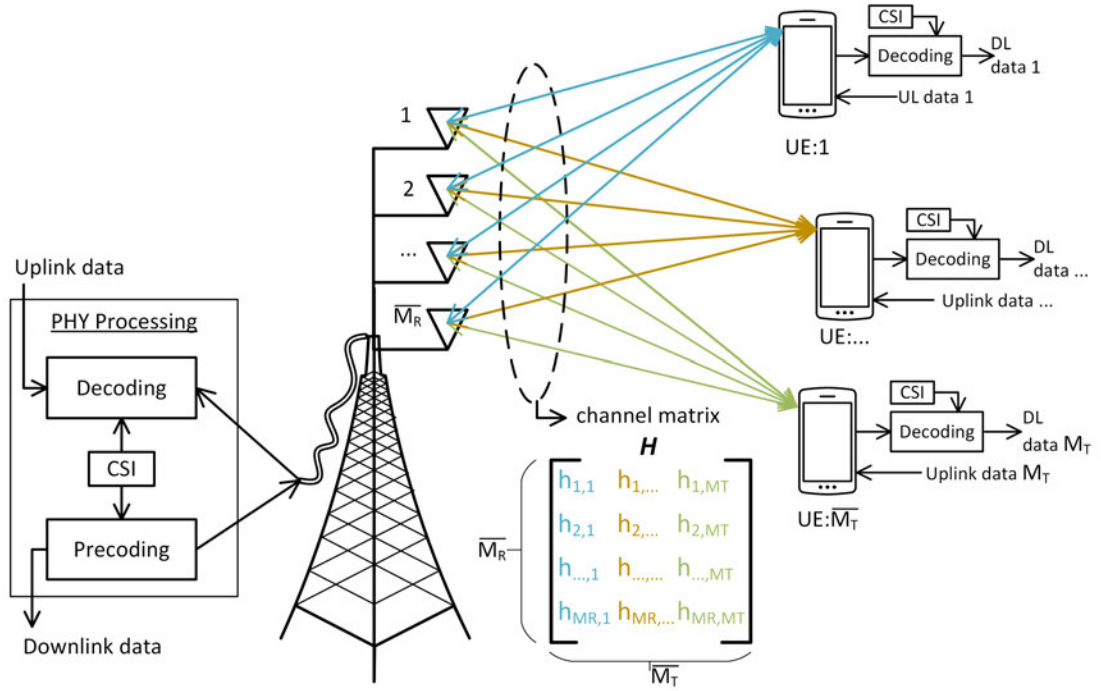


Figure 2.4: Conceptual illustration of massive MIMO loosely based on [2]

Duplexing, Channel Reciprocity, and Channel State Information (CSI) Measurement

The frequency bands in 5G-NR are specified for one of two duplexing modes, namely Time Division Duplex (TDD) or Frequency Division Multiplexing (FDD) [42].

In FDD, two dedicated frequency bands are used for uplink and downlink, respectively. The base station sends Channel State Information - Reference Signals (CSI-RS) in the downlink. Based on this, the UE can assess the quality of the channel and report an index of a codebook in the uplink to suggest a precoding matrix. This is to avoid sending back an excessive amount of channel information in the uplink as each of the M_T UEs would have to report back M_R channel measurements. The CSI-RS is common to all users in the cell. In addition, Demodulation Reference Signals (DMRS) are sent per user to perform channel estimation and signal detection in the UEs. This two-stage scheme reduces the feedback overhead while providing good spectral efficiency [43].

The orthogonal pilots in the uplink sent by the UE are named Sounding Reference Signals (SRS) in 5G-NR terminology. This information does not need to be relayed back to the UE as the massive MIMO detection process makes use of this CSI, and no precoding is required in the UE. A significant advantage for the spatial multiplexing massive MIMO technology is that MIMO processing is not required by the often power-constrained UEs. Especially for configurations with a large number of base station antennas, high user mobility, and high operating frequencies (FR2), the amount of required CSI feedback if such is provided might become too large in FDD mode.

In TDD mode, the same frequency band is used for uplink and downlink operation. It is typically assumed that the channel behaves similarly for uplink and downlink, i.e. a reciprocal channel. This significantly reduces the channel sounding overhead, as only $\overline{M_T}$ SRS symbols are needed in the uplink. However, channel reciprocity might be impacted by RF impairments as the components of the transmit- and receive-chain are not identical in practice [44]. Further, in a scenario where each UE is equipped with multiple antennas, it might be advantageous to use a different number of antennas for reception and transmission. The number of transmit antennas might be reduced to reduce transmitter complexity and thereby save power [45]. TDD is typically the preferred mode of operation for the above reasons; however, with a peak power constraint and in a noise-limited scenario, FDD might deliver a $3\,\mathrm{dB}$ higher SNR [2]. This is simply because in FDD, the transmitter is continuously on, while in TDD, it is not used during reception.

2.3.2 Conventional Signal Detection

Signal detection describes the process of recovering the transmitted symbol in the receiver with the help of CSI. Signal detection is performed in the base station in the uplink case and in the UE in the downlink case. As mentioned above, **precoding** is the process of beam steering in the digital base station, which allows for massive MIMO benefits such as spatial multiplexing. **Beamforming** describes the same process; however, it is often applied to

systems which rely on analogue components as part of the steering process. A typical signal processing chain in a massive MIMO base station with OFDM processing is shown in the block diagram of Fig. 2.5. The MIMO detector is highlighted in light green and supported by the channel estimation block. On the right-hand side, the per-antenna processing steps are depicted schematically. This includes the Analog Front End (AFE), the Digital Front End (DFE), and OFDM processing. The OFDM block is similar as described in Section 2.2 and typically consists of CP removal and an FFT. In the central processing unit, each channel estimation and MIMO detection is performed on a per-subcarrier basis. When evaluating a deployed detector's computational complexity, this also needs to be considered. After MIMO detection, the per-user processes can be started. Therefore, the MIMO detector task can be seen as a mapping from the per-antenna domain to a per-user domain.

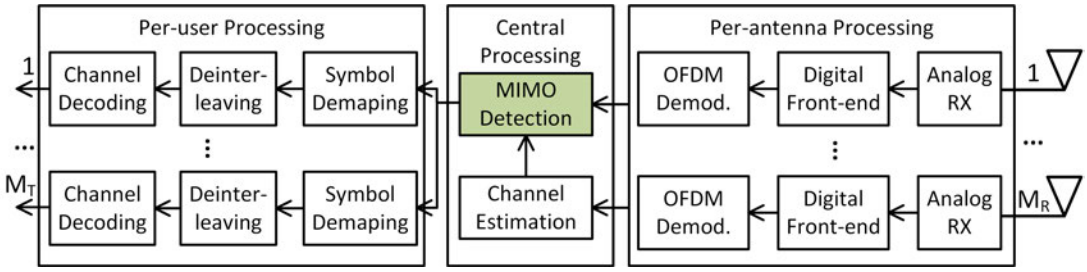


Figure 2.5: Signal processing chain in the massive MIMO base station with OFDM processing [3]

The detection problem can be formulated as a problem in bayesian statistics:

$$\widehat{\boldsymbol{x}}_{MAP} = \underset{\boldsymbol{x}\in\mathcal{C}_C^{M_T}}{\arg\max}\ p(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{H}) = \underset{\boldsymbol{x}\in\mathcal{C}_C^{M_T}}{\arg\max}\ \frac{p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{H})p(\boldsymbol{x})}{p(\boldsymbol{y})}, \tag{2.13}$$

where $p(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{H})$ is the posterior probability, and $p(\boldsymbol{x})$ is the prior. Since the value of $p(\boldsymbol{y})$ is constant over the support of $\boldsymbol{x}$, it does not influence the output of the argmax-function. Eq. (2.13) can therefore be rewritten as

$$\widehat{\boldsymbol{x}}_{MAP} = \underset{\boldsymbol{x}\in\mathcal{C}_C^{M_T}}{\arg\max}\ p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{H})p(\boldsymbol{x}). \tag{2.14}$$

A reasonable assumption in practice is that all values of $\boldsymbol{x}$ are equiprobable, i.e. the Probability Density Function (PDF) of $\boldsymbol{x}$ is uniform over its support. This leads to the Maximum Likelihood (ML) detector:

$$\widehat{\boldsymbol{x}}_{ML} = \underset{\boldsymbol{x}\in\mathcal{C}_C^{M_T}}{\arg\max}\ p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{H})) = \underset{\boldsymbol{x}\in\mathcal{C}_C^{M_T}}{\arg\min}\ \|\boldsymbol{y}-\boldsymbol{H}\boldsymbol{x}\|_2^2 \tag{2.15}$$

Unfortunately, the computational complexity of a brute-force calculation of $\widehat{\boldsymbol{x}}_{ML}$ is prohibitive even for small system configurations as the number of possible arguments in the argmin-term is $(M_C)^{M_T}$.

Precoding and Signal Detection in 5G-NR

Since release 15, the 3rd Generation Partnership Project (3GPP) 5G-NR standards specify two supported codebook-based approaches. A codebook refers in this context to a predetermined precoding matrix used for the beamforming in the base station (Section 2.3.1 introduces precoding). The Type-I codebook approach targets a Single User MIMO (SU-MIMO) scenario where each time-frequency resource is scheduled for a single user [46]. The UE can indicate which precoding matrix it prefers via the Precoder-Matrix Indicator (PMI). Typically, a large number of parallel data streams are provided via spatial multiplexing to the single UE. Each independent data stream is also called a layer in 5G-NR terminology. On the other hand, a Type-II codebook targets a Multi User MIMO (MU-MIMO) scenario in which multiple users receive a smaller number of layers per user on the same time-frequency resource. The precoder indication in both types can only be understood as a recommendation by the UE. The actually used precoder can be freely determined by the BS [32]. At the time of writing, 3GPP Release 18 (5G-NR advanced) is under discussion. An overview of the activities, including massive MIMO, can be found in [46, 47].

Linear Detection

Linear detection is characterised by the goal of finding a matrix $\boldsymbol{W}$ which is multiplied with the received vector $\boldsymbol{y}$ in order to estimate $\widehat{\boldsymbol{x}}$:

$$\widehat{\boldsymbol{x}} = \boldsymbol{W}\boldsymbol{y} = \boldsymbol{W}\left(\boldsymbol{H}\boldsymbol{x} + \boldsymbol{n}\right) = \boldsymbol{W}\boldsymbol{H}\boldsymbol{x} + \boldsymbol{W}\boldsymbol{n}; \tag{2.16}$$

In the simplest case, $\boldsymbol{W}_{MRC}$ can be set to $\boldsymbol{H}^H$ to implement the **Maximum Ratio Combiner (MRC)**. This method is also called matched filtering. The diagonal elements of the Gram matrix $\boldsymbol{G}_T = \boldsymbol{H}^H\boldsymbol{H}$ hold the channel transmit power and phase shift. As the name suggests, MRC estimates $\widehat{\boldsymbol{x}}$ by scaling the transmit symbols according to their power by compensating for phase shifts. This method is sub-optimal if more than one UE is connected, as it cannot eliminate inter-user interference.

On the other hand, the **ZF** detector eliminates inter-user interference completely. The ZF detector can be derived for the massive MIMO case similar to Eq. (B.6) to Eq. (B.8). Alternatively, it can be derived by multiplying $\boldsymbol{H}^H$ to the system equation Eq. (2.12), thereby forming the invertible Gram matrix in the term containing $\boldsymbol{x}$.

$$\begin{aligned} \boldsymbol{H}^H\boldsymbol{y} &= \boldsymbol{H}^H\boldsymbol{H}\boldsymbol{x} + \boldsymbol{H}^H\boldsymbol{n} = \boldsymbol{G}_T\boldsymbol{x} + \boldsymbol{H}^H, \\ \boldsymbol{G}_T^{-1}\boldsymbol{H}^H\boldsymbol{y} &= \boldsymbol{x} + \boldsymbol{G}_T^{-1}\boldsymbol{H}^H\boldsymbol{n} \end{aligned} \tag{2.17}$$

The matrix $\boldsymbol{G}_T^{-1}\boldsymbol{H}^H$ is commonly known as the Moore–Penrose pseudoinverse. By setting $\boldsymbol{W}_{ZF} = (\boldsymbol{H}^H\boldsymbol{H})^{-1}\boldsymbol{H}^H$ the ZF detector is implemented. Although inter-user interference is eliminated, the noise term is not considered for detection. In fact, the noise can be amplified by this processing which degrades the performance in low SNR conditions. This motivates the **MMSE** approach,

which can trade off interference cancellation and noise amplification. Taking the noise into account, the detector finds the matrix $\boldsymbol{W}_{MMSE}$, which minimises the mean of the square error between $\boldsymbol{W}\boldsymbol{y}$ and $\boldsymbol{x}$. From Eq. (B.12), following [48, chapter 2], the MMSE detection matrix can be derived as

$$\begin{aligned}\boldsymbol{W}_{MMSE} &= \mathbb{E}\left\{\boldsymbol{x}\boldsymbol{y}^H\right\}\mathbb{E}\left\{(\boldsymbol{y}\boldsymbol{y}^H)\right\}^{-1}, \\ &= (E_s\boldsymbol{I_T})\boldsymbol{H}^H(E_s\boldsymbol{I_T})^{-1}\left(\boldsymbol{H}\boldsymbol{H}^H + \boldsymbol{I_R}\sigma^2 E_s^{-1}\right)^{-1}, \\ &= \boldsymbol{H}^H\left(\boldsymbol{H}\boldsymbol{H}^H + \boldsymbol{I}_R\frac{\sigma^2}{E_s}\right)^{-1}.\end{aligned} \tag{2.18}$$

The expected value of the first term in the first line in Eq. (2.18) is

$$\begin{aligned}\mathbb{E}\left\{\boldsymbol{x}\boldsymbol{y}^H\right\} &= \mathbb{E}\left\{\boldsymbol{x}(\boldsymbol{H}\boldsymbol{x}+\boldsymbol{n})^H\right\}, \\ &= \underbrace{\mathbb{E}\left\{\boldsymbol{x}\boldsymbol{x}^H\right\}}_{E_s\boldsymbol{I_T}}\boldsymbol{H}^H + \underbrace{\mathbb{E}\left\{\boldsymbol{x}\boldsymbol{n}^H\right\}}_{0}, \\ &= (E_s\boldsymbol{I_T})\boldsymbol{H}^H.\end{aligned} \tag{2.19}$$

As the entries of $\boldsymbol{x}$ are assumed to be independent of each other, the auto-correlation matrix is diagonal with the mean transmit power of the symbol constellation E_s on the diagonal. There is no correlation between the transmit symbol and the noise vector. The expectation value of the second term in line one of Eq. (B.12) is

$$\begin{aligned}\mathbb{E}\left\{\boldsymbol{y}\boldsymbol{y}^H\right\} &= \mathbb{E}\left\{(\boldsymbol{H}\boldsymbol{x}+\boldsymbol{n})(\boldsymbol{H}\boldsymbol{x}+\boldsymbol{n})^H\right\}, \\ &= \boldsymbol{H}\underbrace{\mathbb{E}\left\{\boldsymbol{x}\boldsymbol{x}^H\right\}}_{E_s\boldsymbol{I_T}}\boldsymbol{H}^H + 2\underbrace{\mathbb{E}\left\{\boldsymbol{n}\boldsymbol{x}^H\right\}}_{0}\boldsymbol{H}^H + \underbrace{\mathbb{E}\left\{\boldsymbol{n}\boldsymbol{n}^H\right\}}_{\boldsymbol{I_R}\sigma^2}, \\ &= (E_s\boldsymbol{I_T})\boldsymbol{H}\boldsymbol{H}^H + \boldsymbol{I_R}\sigma^2.\end{aligned} \tag{2.20}$$

Since the noise vector $\boldsymbol{n}$ is drawn from a Gaussian distribution with noise variance σ^2, the expectation value of the covariance matrix has σ^2 on its diagonal and is zero elsewhere. As it will be used later, it is worth noting the identity

$$\boldsymbol{H}^H \left(\boldsymbol{H}\boldsymbol{H}^H + \boldsymbol{I}_R \frac{\sigma^2}{E_s} \right)^{-1} = \left(\boldsymbol{H}^H\boldsymbol{H} + \boldsymbol{I_T} \frac{\sigma^2}{E_s} \right)^{-1} \boldsymbol{H}^H. \tag{2.21}$$

From a computational complexity perspective, the right-hand-side form is preferred as the number of multiplications for $(\boldsymbol{H}^H\boldsymbol{H})\boldsymbol{H}^H$ is $2M_R M_T^2$ while for $(\boldsymbol{H}\boldsymbol{H}^H)\boldsymbol{H}^H$ it is $2M_R^2 M_T$. This difference is significant for $M_R >> M_T$, as in the massive MIMO case.

Due to the law of large numbers, the linear detection algorithms can theoretically perform near to optimum if the number of base station antennas is very large [49, chapter 3.3]. However, in practice, the number of base station antennas is limited, and realistic channels might be challenging to handle for linear detectors. Further, the linear ZF and MMSE detectors require a matrix inverse per channel realisation. As the system size and, in particular, the number of connected users grows, this can become infeasible. Inverting these matrices directly is usually not required. The inverse can be approximated with a Neumann series or polynomial expansion. Alternatively, it can also be decomposed via a modified QR-decomposition or Cholesky-decomposition, which then allows the set up of a signal detection process based on the decomposed matrix products [3]. Although this processing helps with the computational complexity, it can only reach the MMSE performance. The investigation of non-linear, potentially iterative detectors is motivated by finding methods which deliver higher performance at a reasonable computational cost. Iterative detectors are introduced in Section 3.2. For a comprehensive survey of massive MIMO detection techniques, the interested reader is referred to [50].

2.4 Machine Learning Fundamentals

The terms **Artificial Intelligence (AI)**, **Machine Learning (ML)**, and **Deep Learning (DL)** are often used interchangeably; however, they describe different things. AI is the broadest term, and one definition could be a computing system capable of performing intelligent actions on a given task. ML is a subset of AI and describes a set of algorithms that can learn from experience (i.e. from data supplied to them). In the 2010s, computing power reached levels which allowed the training of deep and complex Artificial Neural Networks (ANN) and the idea of Deep Learning (DL) emerged. DL has become one of the most successful and ubiquitous ML techniques. Since the networks contain many layers, they tend to be very computationally and memory intensive.

This work mainly uses the term ML, as it provides the right breadth of the topics discussed without being too specific or general. ML has been applied to a multitude of tasks, such as image recognition, natural language processing, machine translation, and many more [51, chapter 5.1]. These tasks can be grouped into ML problem classes. Well-known problem classes are classification, regression, and synthesis. The detection process in a communication system can also be formulated as a classification or regression problem and is discussed next.

2.4.1 Components of an ML Algorithm

The simplest model of a communication system may be given by

$$y = g(x);\ x \in \mathcal{C}, \tag{2.22}$$

where x is the transmit data at the transmitter and $g(\cdot)$ is the channel which might consist of linear alterations (e.g. channel gains), non-linear alterations (e.g. power amplifiers, etc.) and noise such as thermal noise, interference, quantisation noise, thermal noise in components etc. The signal y is what the receiver obtains. The signal detector can then be given by

$$\widehat{x} = f(y, \theta), \tag{2.23}$$

where $\widehat{x}$ is the reconstruction of x based on the received signal y and the detector parameters θ. As with most machine learning algorithms, the detector task in Eq. (2.23) can also be analysed by describing its fundamental components, namely its dataset, model, cost function, and optimisation procedure [51, chapter 5.10].

Dataset

By the above definition, ML algorithms can learn from experience. The required experience is provided to the algorithm through training and test data. In this work, only supervised learning approaches are considered - even though it might be possible to set up the detection problem as an unsupervised learning method. In supervised learning, along with the training inputs, a ground truth label is also provided to the algorithm. In the above example, the training data might be a collection of y values with associated x labels.

The amount of available training data is limited in many applications. For example, a training dataset for image recognition might have to be labelled manually. The detection task in communications typically does not face this limitation. If the communication system is modelled well, an infinite amount of training- and test-data can be generated as desired. Section 4.2.5 elaborates on the importance of channel models in the massive MIMO case. Further data can be collected with every channel usage during the system's deployment. The availability of training data is an advantage in developing ML detectors.

Model

The function $f(y, \theta)$ in Eq. (2.23) can be interpreted as a ML detector model, where θ are the model parameters. The goal of the ML detector is to achieve high detection performance (i.e. few detection errors) at low cost in terms of computational complexity during deployment. This performance increase is typically realised by one of two alternative design paradigms: data-driven and model-driven designs.

In the **data-driven** design, little or no knowledge of the detection problem is preset in the detector before training. Common DL structures such as ANNs, Convolutional Neural Networks (CNN), and Recursive Neural Networks (RNN) are typically used as building blocks to implement the detector. Then, with a vast amount of training data, the detector learns the detection task. The universal approximation theorem of ANNs with hidden layers and non-linearities states that any continuous function on a closed subset of $\mathbb{R}^N$ can be approximated with an arbitrarily small error by a neural network given enough hidden neurons [51, chapter 6.4.1],[52]. Examples of data-driven detectors can be found in [16, 53]. The data-driven approach might be appealing conceptually; however, it often suffers from high computational complexity.

Model-driven designs avoid this high complexity by building on the established results in communication science. Existing algorithms are extended by ML elements. Most commonly, some parameters of existing algorithms are optimised via ML, leading to higher detection performance. A model-driven design example can be found in [54].

Cost Function

The ground truth label is provided to the detector during training in supervised learning, and the cost function is defined by the **loss** and the model. In the simple example above, the loss can be defined as

$$J(x, \widehat{x}, \theta) = \mathrm{loss}(x, \widehat{x}) = \mathrm{loss}(x, f(y, \theta)) = \mathrm{loss}(x, f(g(x), \theta)). \tag{2.24}$$

A common and simple loss function is the Mean Square Error (MSE) defined as

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(x - \widehat{x})^2. \tag{2.25}$$

Other terms might be added to the loss function to penalise certain properties or implement constraints. This is called regularisation. The process of obtaining the loss is often called **forward propagation**, as the inputs are passed to the network, and each layer computes its results successively.

Optimisation

The target of the optimisation is to minimise the loss

$$\widehat{x} = \arg\min_{\theta} J(x, \widehat{x}, \theta)\ . \tag{2.26}$$

Many optimisation algorithms have been proposed, and optimisation is an entire academic discipline. To illustrate the optimisation concept, the simplest version of an iterative gradient descent algorithm can be expressed as

$$\theta_{i+1} = \theta_i - \epsilon\nabla_\theta J(x, \widehat{x}, \theta_i) = \begin{bmatrix} \theta_{i,1} \\ \theta_{i,\dots} \\ \theta_{i,N} \end{bmatrix} - \epsilon \begin{bmatrix} \frac{\partial J(x,\widehat{x},\theta_i)}{\partial \theta_{i,1}} \\ \frac{\partial J(x,\widehat{x},\theta_i)}{\partial \theta_{i,\dots}} \\ \frac{\partial J(x,\widehat{x},\theta_i)}{\partial \theta_{i,N}} \end{bmatrix}, \tag{2.27}$$

where ϵ is the step size, N is the number of model parameters, and the nabla operator (∇) takes the partial derivatives of $J(\cdot)$ with respect to the current model parameters θ_i. The term **back-propagation** is often used to describe the process of the loss being propagated back in the model to compute the gradient with respect to the model parameters. In practice, the optimisation algorithm might have stochastic elements and might be more complex. A recently popular optimiser is Alternating Direction Method of Multipliers (ADMM) [55].

2.4.2 Artificial Neural Network (ANN)

ANNs have been used in many applications. Neurons are the fundamental computational unit in ANNs. The output of a simple neuron can be written as

$$out = \Phi\left(\sum_{i=1}^{N}(\boldsymbol{k}_i \times \boldsymbol{in}_i) + b\right). \tag{2.28}$$

Where $\boldsymbol{k} \in \mathbb{R}^N$ is the weight vector, $\boldsymbol{in} \in \mathbb{R}^N$ is the input vector, and b and $out \in \mathbb{R}$ are the bias value and the neuron's output respectively. The activation function Φ calculates the neuron's output and is often a non-linear function. Common activation functions are Rectified Linear Unit (ReLU) and Sigmoid functions [56]. Multiple neurons can be connected to the same inputs. Such a grouping is often referred to as a layer, which can be expressed as follows

$$\boldsymbol{out} = \Phi(\boldsymbol{K} \times \boldsymbol{in} + \boldsymbol{b}), \tag{2.29}$$

where $\boldsymbol{K} \in \mathbb{R}^{M,N}$ is the weight matrix, $\boldsymbol{b} \in \mathbb{R}^M$ is a vector and the output $\boldsymbol{out} \in \mathbb{R}^M$ is the output of all neurons in the layer. The number of neurons in the layer is M and each is connected to N inputs. In a fully connected network, each layer's inputs are connected to the previous layer's outputs or the network's input in the case of the first layer. The last layer's outputs are the neural network's output. For a more thorough introduction to machine learning, the interested reader is referred to [51, 56].

2.5 The Need for Hardware Acceleration

Custom digital hardware development is an expensive and time-consuming process. Hardware architecture design, digital circuit implementation, digital circuit verification, and physical implementation on the chip via modern Very Large Scale Integration (VLSI) tools all need to be considered. For this reason, a clear need for custom hardware design must be shown to justify this approach. The following example lays out approximate requirements for signal detection in 5G-NR systems.

The symbol duration in LTE is $\approx 66.67\,\mu s$ [19] and for 5G-NR it can be as low as $\approx 4.17\,\mu s$ [32, Chapter 5]. The number of data-carrying SCs in LTE is up to 1200, and in 5G-NR, it is up to 3300 with a limitation of maximum $400\,MHz$ of total bandwidth. The 3GPP technical specification TR 38.913 defines the target for user plane latency depending on the use case. For enhanced mobile broadband (eMBB), this latency is defined as 4 ms in the downlink and 4 ms in the uplink. However, the processing delay for the base station is just one part contributing to the overall delay, and the standards do not provide delay requirements for sub-components such as detector circuits. In the 5G-NR numerology, one slot consists of 14 symbols in the case of regular cyclic prefix length [35]. The target delay for the base station processing is one slot, according to [57]. The duration of one slot depends on the SCS and ranges from $62.5\,\mu s$ for a SCS of $240\,kHz$ to $1\,ms$ for a SCS of $15\,kHz$.

Central Processing Unit (CPU)

Table 2.1 provides an estimate of the maximum number of required signal detections per second for various 5G-NR OFDM numerologies. The number is between 49.5 and 396 million detections per second. In this very abstract case, no distinction is made between the uplink and downlink scenarios. The last row of the table shows for reference the number of clock cycles a scalar, $1\,GHz$ Central Processing Unit (CPU) would have available per symbol detection. In this simple model, the CPU is assumed to be ideal, meaning one clock cycle equals one executed instruction without consideration of any effects such as

Table 2.1: Influence of selected 5G-NR numerology maximum number of required detections per second. The number of clock cycles in an ideal, scalar CPU per detection is very small.

Numerology selection constant μ	0	1	2	3	4
Subcarrier Spacing $SCS = 2^{\mu} \times 15$ kHz	15	30	60	120	240
Symbol Time [µs]	66.67	33.33	16.67	8.33	4.17
Maximum Bandwidth [MHz]	50	100	200	400	400
Mazimum Number of SC	3300	3300	3300	3300	1650
Detections per second [Millions]	49.5	99	198	396	396
Number of clock cycles available per detection in an ideal, scalar 1 GHz processor	20.2	10.1	5.1	2.5	2.5

memory access, branching, etc. It shows how only a few clock cycles are available in that model. In particular, for massive MIMO, this number of clock cycles is entirely infeasible. Even using super-scalar CPUs, multi-core CPUs, and higher clock frequencies will not provide sufficient processing power.

Graphics Processing Unit (GPU)

GPUs are an alternative to CPUs, which can provide a much higher number of parallel computations. The number of peak operations per second in modern GPUs is in the order of 10^{12} operations. For example, Nvidia's current flagship A100 tensor core GPU is marketed to provide up to 1248 Terra Operations Per Second (TOPS) with 8-bit fixed point precision [58]. These are ideal numbers; in practice, this throughput can typically not be reached. However much throughput GPUs provide, the latency is typically in the order of a few milliseconds at best. For many applications, this is acceptable, such as for AI training or picture classification. A fundamental design principle of GPUs is that multiple threads run on one so-called "Streaming" Multiprocessor (SM) (i.e. one physical execution unit). Even though entire memory hierarchies are implemented, often, these threads need to access the main memory. This introduces considerable latency. However, since multiple threads are scheduled on the same physical resource, the SM can switch to a different thread while the data is retrieved. This is called latency hiding, and it allows achieving high throughput even though access to the main memory takes time [59]. The downside of this approach is high latency.

The work in [60] implements a classical massive MIMO detector on two $1\,\mathrm{GHz}$ Nvidia GTX 980Ti graphics cards. Among others, a conjugate gradient descent algorithm is implemented. It consists of calculating $\boldsymbol{y}_{MRC}$, $\boldsymbol{G}_T$, the MMSE matrix, and the conjugate gradient solver. Results are shown for 128 subcarriers in a system configuration of 128x16 antennas and a modulation type of QAM16. The sum throughput on both GPUs is $286.57\,\mathrm{Mbit/s}$ with $3.66\,\mathrm{ms}$ latency. This is an impressively low detection latency for GPU processing; however, assuming a SCS of $15\,\mathrm{kHz}$, the latency is ≈ 55 symbols. For a SCS of $240\,\mathrm{kHz}$, the latency is ≈ 878 symbols. This is for a relatively small number of users, with only 128 SC and making use of two GPUs.

Similar limitations can be expected in ML-specialised chips such as Tensor Processing Units (TPU). On top of that, the processor is specialised to ML workloads only, which might make it difficult to implement other workloads such as model-driven detectors.

Application Specific Integrated Circuit (ASIC)

Standard-cell ASICs are the best choice from a throughput, performance, and power perspective. Further, the digital logic, memory hierarchy, and interface can be freely designed. The downside is, of course, the enormous engineering effort and production cost. This is often an uneconomic and unnecessary commitment for prototyping experimental, scientific digital circuits.

Field Programmable Gate Array (FPGA)

A much more suitable platform for prototyping are FPGAs. They provide a good trade-off between processing power and flexibility as they allow custom logic implementation. In contrast to ASIC implementations, the digital design is not implemented directly in transistor-level circuits. Instead, there are a number of predefined primitives implemented on an FPGA chip which can be "programmed" to perform the desired logic function. These primitives themselves are instead implemented via transistor-level circuits in an ASIC design flow. The user design is brought to the FPGA in an Register Transfer Level (RTL)

design flow. The disadvantage of FPGAs compared to ASICs is the overhead of the configurability. The number of physical transistor-level resources for a given logical functionality is much higher in the FPGA. This leads to higher silicon area and cost, higher power, and lower speed. However, these are often acceptable disadvantages to supporting rapid prototyping and benchmarking for scientific purposes.

FPGAs have another well-established field of application, namely, where high processing power is needed, while the number of deployed units is low. Also, the need for in-the-field updates once deployed is a driver for using FPGAs. In this case, ASIC development is especially uneconomical or unpractical. The deployment in 5G-NR base stations is such an application. This also supports targeting FPGAs in this project.

FPGAs allow the user to define large parts of the memory hierarchy and processing pipeline. This enables low detection latency compared to GPUs while maintaining high throughput. The recent work in [61] compares Nvidia's T4 12 nm GPU with Intel's 14 nm, AI-optimized Stratix 10 NX FPGA. The authors claim a 24x compute speedup on the FPGA in their ML workload. Also, a system-level performance comparison shows that the system latency in the FPGA is 10x smaller thanks to the 100 Gbps Ethernet interface.

A more detailed introduction to FPGA technology is presented in Appendix B.2. The FPGAs architecture, design primitives and a basic design flow are discussed.

2.5.1 Targeted platform and Circuit Metrics

For evaluation of the digital circuits presented in this thesis, Alpha Data's ADM-XRC-9R1 System on Module (SoM) featuring the Xilinx XCZU-27DR-2-FFVE1156 RFSoC is targeted [7, 62]. The Xilinx RFSoC features eight 12-bit and 4 Giga Samples Per Second (GSPS) ADC blocks. Each block features hardened fil-

Table 2.2: Resources provided by the FPGAs in Alpha Data's ADM-XRC-9R1 board [7].

	RFSoC XCZU27DR [62]
Device Family	Ultrascale+ (16nm)
LUTs	425k
FFs	850k
DSP slices	4272
BRAM	1080 (38 Mb)
URAM	80 (22.5 Mb)
Distinguishing feature	RF chains, 4x ARM A53, 2x ARM R5

tering, mixing, and downsampling circuits, allowing for direct sampling in the microwave bands. Similarly, eight 14-bit 6.554 GSPS DAC blocks are featured. This platform is highly suitable for 5G-NR base station deployment, among other applications. Its available resources are summarised in Table 2.2.

The usage of **FPGA primitives** is one of the main circuit metrics. The measurement is typically provided in the number of used Lookup Tables (LUT), Flip-Flops (FF), Digital Signal Processors (DSP), Block RAMs (BRAM), and Ultra RAMs (URAM).

Regarding circuit performance, **latency and interval** are core metrics. The latency describes the time from when all inputs are fully available to a circuit until it has produced all outputs completely. In synchronous circuit design, pipelining is an important technique to reduce the circuit's critical path length and thereby allow high clock frequencies. This leads to long latency; however, once the pipeline is filled with data, a new output can be produced at the interval of the longest pipeline stage. We follow Xilinx terminology and name this delay the interval of a circuit. The reciprocal of the interval is the **throughput** in outputs per second.

Fixed-point numbers are useful in Xilinx FPGAs as they can be implemented efficiently. Background information on fixed point numbers can be found in Appendix B.2.3. This work used the following notation as a convenient way to indicate the fixed point format: S IL.FL. If a number is unsigned, the signed bit is not needed, and the format is indicated as IL.FL. For example, a signed fixed-point number with a total of 12 bits, six integer bits, and five fractional bits

is indicated as S6.5. An unsigned fixed-point number with 12 integer bits and three fractional bits is 12.3. The **Most-Significant Bit (MSB)** marks the bit with the largest contribution to the binary number. The **Least-Significant Bit (LSB)** is the bit with the smallest contribution in the binary number.

Power consumption is not reported in this thesis since the power numbers highly depend on the placement and routing of a block. Without the surrounding blocks and the system level FPGA design, the placement and routing will vary, and accurate power numbers will be challenging to obtain. Approximate, high-level power estimation is possible and scales approximately with the number of resources used. To avoid providing misleading numbers, we forego reporting these approximations. Also, much work in the literature to which we compare our designs does not provide these numbers, giving our power numbers limited expressiveness.

2.6 Summery

This chapter provided background information aiding the understanding of the rest of the thesis. First, fundamental principles in communication and the task of signal detection were introduced. Then, the OFDM and massive MIMO detectors with the corresponding system models were discussed. The high computing requirements for real-time processing of detection in LTE and 5G-NR were laid out in Section 2.5. Based on this, FPGAs have been found to be the most suitable platform as they offer the benefits of custom logic design at a reasonable cost.

The following chapters use this background information to develop real-time capable neural network detectors from ML algorithm design to digital circuit implementation. In the following Chapter 3, we will develop one of the first real-time neural network OFDM detectors. Algorithmic improvements of the neural network, neural network simplifications, accelerator design considerations, and finally, an FPGA implementation with in-depth profiling follow.

Chapter 3

Machine Learning enhanced, Real-time OFDM Signal Detection

3.1 Introduction

One modulation technique of high interest is Orthogonal Frequency-Division Multiplexing (OFDM), as it is used in many communication systems such as in the downlink of LTE [19], in the downlink of 5G-NR, and optionally in the uplink of 5G-NR [20]. Various proposals to perform machine learning-based channel estimation and signal detection in an OFDM system have been made, showing increased detection performance as compared to traditional methods. However, real-time processing is rarely considered and will be at the centre of the investigation in this chapter. The presented research ranges from algorithmic improvements to digital circuit design.

An overview of the state-of-the-art ML detectors for OFDM is provided in Section 3.2. A promising proposal for a joint OFDM channel estimator and signal detector is presented in [16]. It is a good example of a data-driven approach, provides good performance gains over conventional algorithms, and its implementation is conceptually simple. Throughout this chapter, we often speak of the signal detector; this implicitly also includes the channel estimation, as the joint approach cannot be separated in the design. The detector is implemented as an ANN.

Despite the algorithmic advancements, the deployment of neural networks in actual receiver hardware has been very limited. In [22], an autoencoder is used in an over-the-air OFDM system. The autoencoder is first trained offline with a mathematical channel model and then deployed via two SDRs. In the deployment stage, the SDRs act as front ends, and the online training and neural network inference are performed on GPUs. However, the inference is not performed in real-time, and the feasibility of the hardware setup for actual real-time deployment is not assessed.

We notice a lack of real-time processing of neural-network-based communication systems in the literature. The here used definition of the term real-time is provided in Section 1.1.2. Real-time processing is challenging as the model complexity of many proposed neural network receivers is infeasibly high. Also, the throughput and latency requirements of communication systems are challenging. In the LTE downlink, the OFDM SC spacing is 15 kHz which leads to a symbol duration of $\approx 66.7\mu s$ [19]. With the new numerologies of up to 240 kHz SC spacing in 5G, the symbol duration can be as low as $\approx 4.17\mu s$ [20]. It is not strictly required to process each symbol within one symbol time; however, latency is a major performance metric of any receiver.

FPGAs are popular for implementing neural network inference due to their programmability, comparatively low power, and cost. They are suitable for prototyping neural network accelerators, as noted in Section 2.5. Many accelerator designs have been proposed in the literature for general-purpose neural networks in recent years. Multiple factors determine the optimal accelerator design, including the neural network type, the chosen loop unrolling strategy, the on-chip and off-chip memory characteristics, the available computing resources, and more. However, a typical accelerator will have common elements such as on-chip buffers, computational units, control logic, host interfaces, and memory interfaces. Surveys on neural network FPGA accelerators can be found in [63] and [64].

3.1.1 Contribution

A novel neural network structure for joint OFDM signal detection and channel estimation in the receiver is proposed in Section 3.3.1. The algorithmic exploration undertaken in this work is loosely based on [16]; however, the new neural network structure significantly reduces the computational complexity and memory requirement, making real-time processing possible.

To further reduce the memory requirement, the methods of deep compression are evaluated and applied to the neural network [65]. Deep compression seeks to reduce a given neural network's memory requirement by applying three techniques: 1) quantisation, 2) pruning, and 3) Huffman coding. Quantisation means to represent weights with a low number of bits while maintaining accuracy (Section 3.3.3). Pruning eliminates small weight values which do not contribute much to the network's output (Section 3.3.4). Huffman coding compresses a stream of weights by taking the probability of each word and assigning shorter codes to more likely occurring words (Section 3.4.1).

Based on the algorithmic improvements and the findings on deep compression, a custom hardware accelerator is designed. This accelerator can process one or more neural network detectors. It supports small batch processing, superscalar Huffman decoding and branching neural networks, making it ideal for the detector workload (Section 3.4 and Section 3.5).

In short and to the best of our knowledge, we implement the first real-time neural network OFDM detector, enabled by:

- Memory reduction by a.) improved neural network structure and b.) compression of the network
- A real-time capable hardware accelerator tailored specifically to the signal detector workload

The developed design could replace the detector and channel estimator blocks in Fig. 2.3 without major changes in the receiver architecture. As the proposed detector does not need to be trained on a specific channel realisation no online training is required. This allows us to simply replace the classical channel

estimator and detector blocks with the proposed design in the receiver. However, long-term channel changes might also necessitate a background training process for the receiver. The investigation of such requirements is required in future work. The training can be detached from the embedding of the signal detector in the reception chain; however, an easy way to change the neural network receiver's weights needs to be provided to facilitate this. As a first step, the here presented work only considers the Intellectual Property (IP) design of the receiver. The integration into a receiver has to be considered in future work.

The reported neural network, accelerator design, and the results are covered by the publication in [27]:

> S. Brennsteiner, T. Arslan, J. Thompson, and A. McCormick, "A Real-Time Deep Learning OFDM Receiver," ACM Transactions on Reconfigurable Technology and Systems, vol. 15, no. 3, pp. 26:1–26:25, Dec. 2021 [Online]. Available: https://doi.org/10.1145/3494049

3.2 State of the Art

One of the first neural network-based OFDM detectors was proposed as an **autoencoder**, capable of inserting pilot symbols as required [66]. The work follows a hybrid approach where neural networks individually encode and decode data for each SC before passing it to traditional OFDM blocks. The transmitter modulates the data via Inverse Discrete Fourier Transformation (IDFT), and a cyclic prefix is added. In the receiver, these steps are reversed, and the decoder neural network is invoked on a SC basis. The insertion of pilots based on requirement is impressive; however, the approach faces scalability issues. First, the computational complexity is very high with $\approx 400k$ parameters per SC. Second, the setup with messages and one hot encoding of these messages is not scalable, and only Quadrature Phase Shift Keying (QPSK) modulation has been tested. An overview of the most influential ML OFDM receivers is

provided in Fig. 3.1. Autoencoders for more general communication systems were among the first proposals of using ML for communication tasks [53]. Over-the-air communication of such a design with programmable radio platforms and GPUs for training has been shown [22].

A receiver-only neural network is presented in [16], which we call **Fully Connected Deep Neural Network (FCDNN)**. The authors present a comb-type OFDM system with fixed positions of pilots across the frequency axis. The channel estimation and signal detection are performed simultaneously in a single neural network for eight SCs. This work incorporates minimal assumptions about the underlying communication system, and the detector is purely based on Fully Connected (FC) layers. The detector shows clear performance improvements compared to classical MMSE detection, particularly under difficult system configurations such as omitted cyclic prefix, clipping, and distortions of the transmit signal.

A model-driven design for OFDM detection was **ComNet** [24]. It is inspired by simple, commonly used linear processing algorithms for channel estimation and signal detection. Namely, LS channel estimation and ZF signal detection. ComNet provides the solutions of these simple detectors as initial values to neural networks, which refine and improve the result. For channel estimation, a simple, fully connected layer is used as a refinement network, while for detection, multiple Bi-directional Long Short-Term Memory (LSTM) neurons in combination with a FC layer are used. ComNet supports QAM64 modulation types, where each output bit is mapped to one neuron with a sigmoid activation function. This leads to the number of output neurons being $N_{SC} \times \log2(\mu^2)$, where μ^2 is the number of constellation points in the modulation scheme.

The work in [67] introduces the **Deep Complex-valued Convolutional Network (DCCN)** architecture for channel estimation and signal detection. As the name suggests, it uses the expressiveness of complex-valued neural networks and relies on convolutional operations. It uses the convolution operations in a natural way to replace classical Discrete Fourier Transformation (DFT) and IDFT operations, thereby making them learnable. Further dense, complex lay-

ers are introduced to implement common tasks such as channel estimation and detection. The architecture outperforms classical Linear Minimum Mean Square Error (LMMSE) estimation and detection techniques in Rayleigh fading channels. The exact number of parameters or required operations per SC is not reported; however, as the network consists of multiple convolutional layers and fully connected layers, the number of operations is certainly considerable.

Another detector called **SwitchNet** is proposed in [67]. Observing a diverging over-the-air performance of ComNet from offline training inspires SwitchNet's architecture. The Channel estimation part of the receiver is initialised with the LS estimate before going into multiple refinement networks. The outputs of refinement networks are weighted by simple scalar parameters before being summed up to the final output. During deployment, only these scalar parameters are trained online. The work provides a detailed comparison between SwitchNet, ComNet, and the DNN of [16] in terms of offline and over-the-air detection performance, computational complexity, and more. The computational complexity of SwitchNet in the considered scenario is $\approx 13\times$ smaller than in [16]. The receiver is accelerated via GPU; however, it is not real-time capable.

The work in [23] uses a different ML concept, namely, **Echo State Network (ESN)** from the domain of reservoir computing. ESNs are capable of processing temporal information and can outperform RNNs in certain tasks. The key idea is to provide a reservoir of states which are initialised at random and have connections with random strengths to each other. Inputs are mapped via input weights to the reservoir, and outputs are generated via output weights from the reservoir. During training, merely the output weights are trained, making for efficient training and low complexity. In [23], a 4x4 OFDM-MIMO system with 64 SC and 100 symbols per frame is evaluated. Detection performance is evaluated offline and compared to classical methods such as LMMSE. Unfortunately, no comparison with any of the above detectors is provided. Next, the design implements a SDR frontend and FPGA backend real-time detector setup. The over-the-air performance is evaluated in multiple Line Of Sight

(LOS) and non-LOS scenarios. The proposed ESN detector outperforms the LMMSE detector by a small margin in all scenarios. The FPGA accelerator details are not reported in detail, presumably for brevity reasons, as the work is already quite exensive.

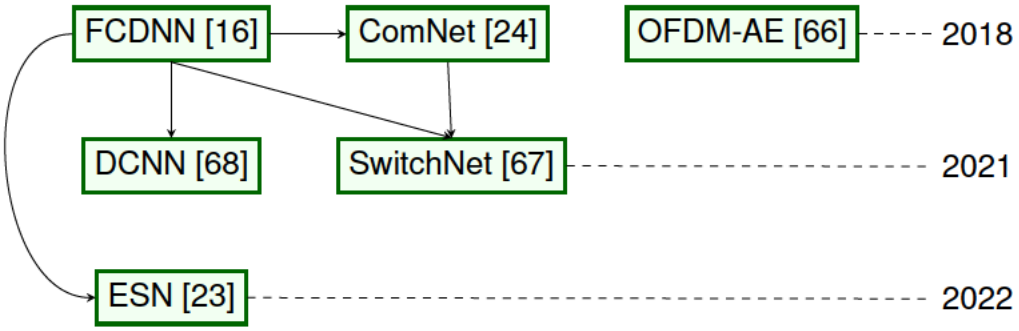


Figure 3.1: Overview of recently proposed ML OFDM detectors, including the most important citations as arrows and the year of publication on the right-hand side.

A survey laying out opportunities and challenges for deep learning in the physical layer can be found in [14]. Yao et al. describe applications in the 5G Radio Access Technology Layer [69]. Broader surveys on machine learning in communications, which also include the physical layer, are presented in [9] and [70].

The remainder of this chapter is loosely based on the work in [16] for various reasons. First, it was one of the most advanced and first neural network detectors at the time of researching the real-time detector approach. Second, it is a purely data-driven detector approach, and it is of interest to optimise and profile such a detector for real-time deployment. Third, it is one of the few works using more advanced channel models (WINNER II in this case), and it shows good detection performance in terms of Symbol Error Rate (SER). Also, the FC layer structure simplifies the development of an FPGA accelerator.

3.3 OFDM Neural Network Detector

A basic OFDM system model has been introduced in Section 2.2.1. In LTE and 5G-NR, symbols are arranged along the time- and frequency axis, forming a resource grid as shown in Fig. 2.1. To acquire channel state information, known reference symbols (i.e. pilot symbols) are typically embedded in equidistant intervals in the resource grid. The receiver uses these reference symbols to estimate the channel and aid symbol detection. The neural network proposed in Subsection 3.3.1 jointly handles these two tasks. For a more detailed introduction to OFDM, channel estimation, and signal detection, we refer the interested reader to the standard literature (e.g. [71]).

3.3.1 Detector Architecture

The receiver signal $\boldsymbol{y} \in \mathbb{C}^{M_{SC}}$ holds one complex-valued entry for each SC as introduced in Section 2.2.1. Since the positions of the pilot symbols are known in the receiver, the channel can be estimated at these SC. Based on this, the channel can be interpolated and estimated at data-bearing SCs. Classical channel estimation approaches are briefly discussed in Section 2.2.2. Classical OFDM signal detectors are reviewed in Section 2.2.3.

Most machine learning frameworks, such as Tensorflow, are based on real-valued calculations. To allow the real-valued neural network to perform detection and channel estimation, the complex-valued receiver signals are converted to a real representation by interleaving real and imaginary values:

$$\mathbf{y_{real}}[2i] := \Re(\boldsymbol{y}[i]);\ i \in \{1, .., M_{SC}\} \tag{3.1}$$

$$\mathbf{y_{real}}[2i+1] := \Im(\boldsymbol{y}[i]);\ i \in \{1, .., M_{SC}\} \tag{3.2}$$

Where $\mathbf{y_{real}} \in \mathbb{R}^{2M_{SC}}$ is the real input vector. The neural network then recovers an estimate of the transmitted symbol

$$\widehat{\boldsymbol{x}} = f_{NN}(\mathbf{y_{real}}). \tag{3.3}$$

The target of the neural network, thus, is to recover the transmit vector $\boldsymbol{x}$ with as few errors as possible. The detector is inspired by the work in [16] and consists of multiple fully connected neural network layers. The detection task only requires neural network inference during deployment. A batch-wise inference algorithm of a fully connected neural network is shown in Algorithm 1. The number of layers is given by the parameter M, and the number of neurons per layer is $[N_1, ..., N_M]$. Further, the number of elements in the input equals $2M_{SC}$, and the activation function is given by $\Phi_m(\cdot)$. Each layer might use a different activation function, which is indicated by the suffix m.

Algorithm 1: Inference in a multi-layered, fully connected neural network considering multiple input samples in a batch

input : Data: multiple $\boldsymbol{x} \in \mathbb{R}^{2M_{SC}}$
input : Weights:
$\boldsymbol{W} = \{\boldsymbol{W_1} \in \mathbb{R}^{2M_{SC},N_1}, \boldsymbol{W_2} \in \mathbb{R}^{N_1,N_2}, ..., \boldsymbol{W_M} \in \mathbb{R}^{N_{M-1},N_M}\}$
input : Bias: $\boldsymbol{B} = \{\boldsymbol{b_1} \in \mathbb{R}^{N_1}, \boldsymbol{b_2} \in \mathbb{R}^{N_2}, ..., \boldsymbol{b_M} \in \mathbb{R}^{N_M}\}$
output: multiple $\boldsymbol{y} \in \mathbb{R}^{N_M}$

$\boldsymbol{Z} = \{\boldsymbol{z_0} \in \mathbb{R}^{2M_{SC}}, \boldsymbol{z_1} \in \mathbb{R}^{N_1}, ..., \boldsymbol{z_M} \in \mathbb{R}^{N_M}\}$; // Intermediate signals
foreach $\boldsymbol{x}$ **do** // Batch Loop
  $\boldsymbol{z_0} \leftarrow \boldsymbol{x}$;
  for $m \leftarrow 1$ **to** M **do** // Layer Loop
    for $n \leftarrow 1$ **to** N_M **do** // Neuron Loop
      $Z_{m,n} \leftarrow 0$;
      for $i \leftarrow 0$ **to** N_{M-1} **do** // Input Loop
        $Z_{m,n} \leftarrow Z_{m,n} + Z_{m-1,i} \times W_{m,n,i}$;
      end
      $Z_{m,n} \leftarrow \Phi_m(Z_{m,n} + B_{m,n})$;
    end
  end
  $\boldsymbol{y} \leftarrow \boldsymbol{Z}_m$;
end

The detector in [16] follows the processing scheme in Algorithm 1 exactly. It receives as input 64 SC and produces the detection result of eight selected SC via five FC layers of size 256, 500, 250, 120, and 16, respectively. For broader system bandwidths, multiple detectors have to be implemented, each with multiple narrow-band reception chains on the system level. This is not ideal, as multiple narrow-band reception chains are required on the system level. Systems with more SC have been tried in our training setup but showed a loss in performance. This is one of the main limitations of the approach and is further discussed in Section 3.7.

Algorithmic Complexity Reduction

Especially for FC-layer-based detectors, the number of neurons in the detector determines its complexity as each neuron increases the number of required weights in its input and output weight matrix. This causes issues as weight storage capacity and transmission throughput are limited. Also, some performance metrics, such as resource usage or throughput, will be impacted. It is, therefore, desirable to reduce the number of required neurons as much as possible.

We propose to share the first two layers of the neural network detector and then branch out into the last two layers for each set of SC, as seen in Fig. 3.2. Each of the M_D detectors processes $M_B \times M_S$ SC. Where M_B is the number of branches in one detector, and M_S is the number of SC processed per branch. An entire neural network must be deployed in [16] for detecting eight SC, leading to many neural network weights. The idea is to allow the first two layers to learn shared features between all the detected SC and to learn individual properties per branch in the last two layers.

To support this hypothesis, we trained a neural network in which the first two layers are split, and multiple independent networks are formed such that the sum overall complexity of the neural network is the same as the proposed branching neural network. In this case, a complete loss of performance is observed. This indicates that the first two layers in the branching neural network

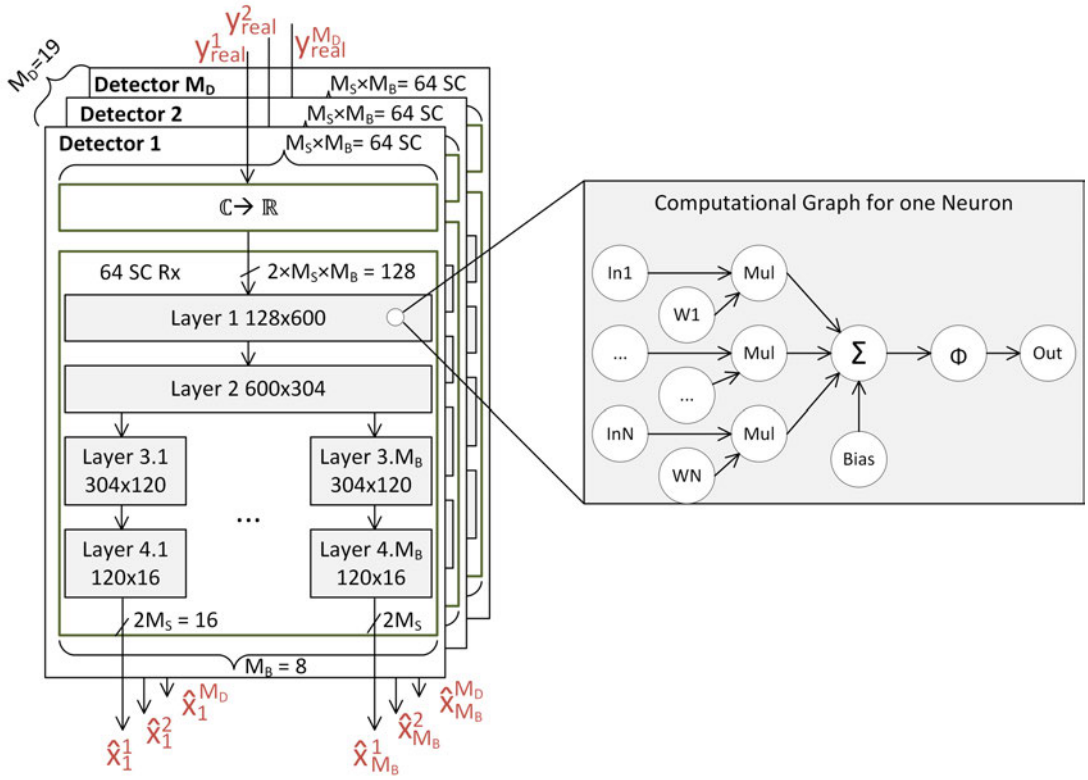


Figure 3.2: Neural network receiver structure consisting of M_D detectors, each consisting of a branching neural network. On the right, the computational graph of one neuron is highlighted.

can indeed extract common features important to the detection in all branches. The branching neural network takes advantage of this observation, and our experiments show similar performance of this architecture and the architecture presented, in [16], while complexity is significantly reduced.

The number of SC per branch is chosen to be eight (denoted by $M_S = 8$), and the number of branches per detector is eight as well (denoted by $M_B = 8$), leading to 64 SC to be processed per detector. Each detector is completely independent of the other detectors and has its own reception chain, as outlined in Fig. 3.2. For the hardware implementation of the receiver, M_D parallel neural network detectors are considered, of which each is processing independent SC. We evaluate our design by choosing M_D to be equal to or less than 19, which gives a maximum total of 1216 SC processed. Not considering the

16 last SC of the 19th detector, this results in 1200 effective SC, and the sum-bandwidth of the occupied channels is 18.015 MHz, including the Direct Current (DC) SC. This means that the 19 detectors can cover all standard bandwidths as specified in LTE.

3.3.2 Training Setup

The resulting graph for each detector has multiple outputs, and for each of the M_B branches, a mean square error loss value is calculated as

$$L_b = \frac{1}{2M_S} \sum_{n=1}^{2M_S} (\widehat{\boldsymbol{x}}_b[n] - \boldsymbol{x}[b \times 2M_S + n])^2 \text{ where,} \tag{3.4}$$

$$\widehat{\boldsymbol{x}}_b[n] = \widehat{\mathbf{x}}[b \times 2B_S + n], \quad n \in \{1, ..., 2M_S\}, \tag{3.5}$$

for $b \in \{1, ..., M_B\}$, and where $\widehat{\boldsymbol{x}}$ is the estimated symbol value (i.e. the neural network output). The symbol initially sent by the transmitter is $\boldsymbol{x}$ (i.e. the training label). The loss functions are summed to calculate the total loss as used in the optimisation algorithm as

$$L_{total} = \sum_{b=1}^{B} L_b. \tag{3.6}$$

We train only one detector for 64 SC for simplicity in our experiments. However, the proposed hardware accelerator can accelerate multiple detectors with distinct trained weights and activations. Results on performance and memory reduction are presented in Subsection 3.6.2.

The channel model used for training is WINNER II and has been taken from [72]. The noise in the channel $\boldsymbol{n}$ is added to each sample and assumed to be AWGN with the standard distribution of $N_0/2$.

The structure of the used OFDM system for training and evaluation is depicted in Fig. 3.3. Some of the source code of the environment has been taken from [72]. However, many alterations have been made ranging from the implementation of pruning and quantisation to updates of the communication blocks, the implementation of new neural network structures, the system setup, and the performance profiling methods.

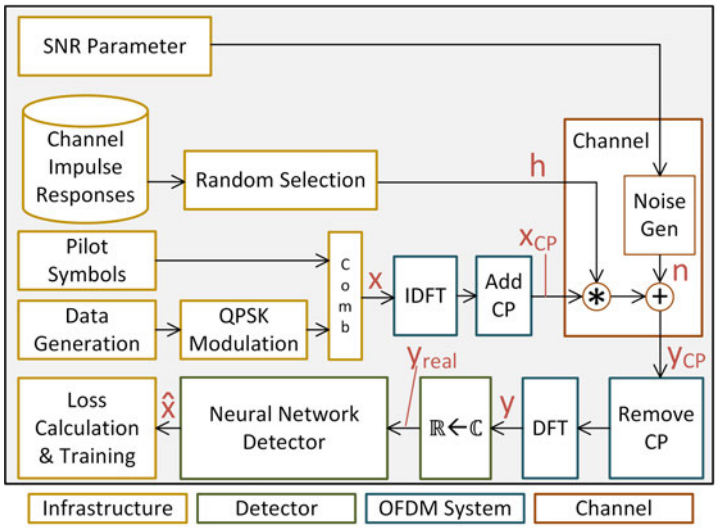


Figure 3.3: Neural Network training and evaluation setup for a single neural network detector. Each detector processing 64 SC has to be trained independently. The implementation is done via Python and Tensorflow.

3.3.3 Quantisation

Changing the floating-point model parameters to fixed-point data types is often referred to as model quantisation. We use the abbreviation Bitwidth Weights (BW) to mean the bit-width for weights and Bitwidth Activations (BA) to mean the bit-width of activations. Two approaches for quantisation are common. Firstly, post-training quantisation takes the parameters after training and applies quantisation to them. This often affects the network's performance, especially when quantising to small bit widths. Secondly, quantisation-aware training applies quantisation during the training process. Our training approach is based on [73], where each parameter is still saved as a floating-point number during training. A quantised version of the parameters is used in the forward pass to

calculate the loss. In the back-propagation step, the floating-point parameters are updated. The floating-point representation of the parameters is required to allow small updates during training. Quantisation results are presented in Subsection 3.6.1

3.3.4 Pruning

Pruning seeks to reduce the number of total weights in a neural network by considering only weights with large magnitudes, as weights with small magnitudes do not contribute much to the output value of a neuron. In our setup, we use the pruning framework provided by google-research [74]. During the building of the computational graph, masking nodes are added, which mask out selected weights in the forward pass during training. The pruning framework accepts a target sparsity parameter which sets a percentage of weights to be masked. In other words, the pruning is not dependent on an absolute threshold value but removes the specified percentage of weights with the lowest magnitudes. The masking can be delayed by an arbitrary number of global training steps. This is useful as an accurate neural network can be trained first before gradually increasing the sparsity until the target sparsity is reached. Using a gradual increase of sparsity makes converging to a good solution more likely. Pruning results are presented in Subsection 3.6.1.

3.4 Accelerator Design Considerations

A well-established model to classify neural network accelerators is the roofline model [75]. It allows classifying the limiting factor in the performance of a design as either memory limited or computationally-limited. The proposed neural network exclusivity consists of fully connected layers, and the number of weights is large. This leads to a memory-limited design. Many considerations in the presented architecture take the memory limitation into account and are measures to mitigate the effect of a memory-limited accelerator. The presented ar-

chitecture is specifically tuned to the task of accelerating the above-introduced OFDM detector neural network, which is also the reason for designing a neural network accelerator instead of relying on a typical abstraction framework (e.g. [76]). The accelerator distinguishes itself by the following main features:

1. Each weight memory interface transfers multiple Huffman encoded data-streams in parallel from memory. The bit-width of the memory interface is shared among multiple weight streams and decoupled via First In First Outs (FIFO), such that each Huffman decoder can work on a small number of bits, reducing the performance requirement per Huffman decoder. Details on the memory interface and the Huffman decoders are described in Section 3.4.1.
2. To make the most efficient use of any memory-transferred weights, each weight is used multiple times on different OFDM symbols. First, multiple OFDM symbols are collected, before they are processed in small batches. Batch processing is a known technique to increase performance in certain inference accelerators [77, 78]. The application in communication systems is a good target for batch processing, as it allows a trade-off between latency and throughput with a given memory performance. Details are described in Section 3.4.2
3. The computational graph of the proposed OFDM-detector branches after the second fully connected layer. Switching between two intermediate result buffers in the accelerator allows efficient storing of intermediate results while processing the remaining layers. Section 3.5.1 expands on this.

The above considerations lead to the following loop-unrolling strategy. The batch level loop in Algorithm 1 is unrolled to the extent that buffering and batch processing of symbols is required. We call this unrolling factor PAR_S. As the second unrolled dimension neuron loop is chosen, such that multiple neuron outputs can be calculated in parallel. This is advantageous as the available memory bandwidth can be easily split up into multiple parallel weight streams and as a single DSP block in the FPGA can be used as a Multiply-ACcumulate (MAC) unit. This parameter is named PAR_N. As the final unrolled dimension,

we chose the innermost loop of input weights. This is to reduce the number of required input FIFOs and Huffman decoders, as one chain of FIFOs and Huffman Decoders can provide up to two weights per clock cycle. This unrolling parameter is named PAR_I.

To showcase the proposed real-time neural network detector architecture, we chose a cutting edge FPGA-platform, namely Alpha Data's RFSoC ADM-XRC-9R1 [7]. The platform provides the Xilinx Zynq Ultrascale+ XCZU27DR-2 FPGA and two 8Gb DDR4-2400 Synchronous Dynamic Random-Access Memory (SDRAM) memory chips connected to the FPGA fabric [62]. Each memory can provide a theoretical throughput of 64 bits of data at a rate of 300MHz. On top of that, the FPGA provides 22.5 Mbit of on-chip URAM which we utilise as weight memory for some detectors. BRAM resources are used for various buffers and for storing activation values.

As will be seen in Section 3.6.1, a quantisation of six bits for the weights and 16 bits for the activations provides a good performance/compression trade-off. Some of the below design considerations assume this quantisation, but may also apply to other quantisation schemes.

3.4.1 Memory Interface and Huffman Decoding

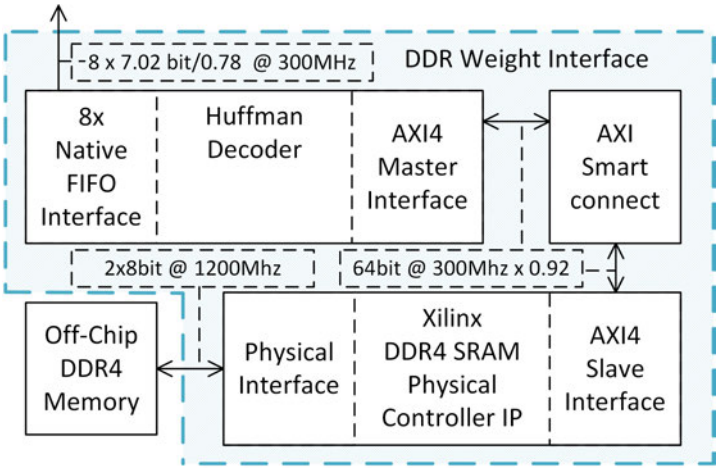
On-chip memory of even large FPGAs is not sufficient to hold all weights for all detectors, making off-chip memory access required. The off-chip memory access is the processing bottleneck in most designs, and the memory interface has to be at the centre of all design considerations. In the chosen example platform, two state-of-the-art memory chips are available. The design has to match the maximum memory performance of 64 bits of data at a rate of $300\,\mathrm{MHz}$ per memory chip as closely as possible. Utilising more parallel memory chips or other memory technology, such as High Bandwidth Memory (HBM) or GDDR5, higher performance could be achieved, but would not show the feasibility of a neural network-based detector on a standard, off-the-shelf platform.

The physical interface to the DDR4 memory is handled by a Xilinx UltraScale Architecture-Based FPGAs Memory IP [79]. It consists of a physical layer, interfacing the DDR4 memory, a memory controller, and a user interface block. The user interface provides a standard Advanced eXtensible Interface (AXI)4 slave interface as shown in Fig. 3.4a. The AXI interface allows a maximum of 256 words to be transferred per burst before another addressing phase has to be performed. This limits the effective throughput. The measured throughput is 256 words per 278 clock cycles or 0.92 words per clock cycle.

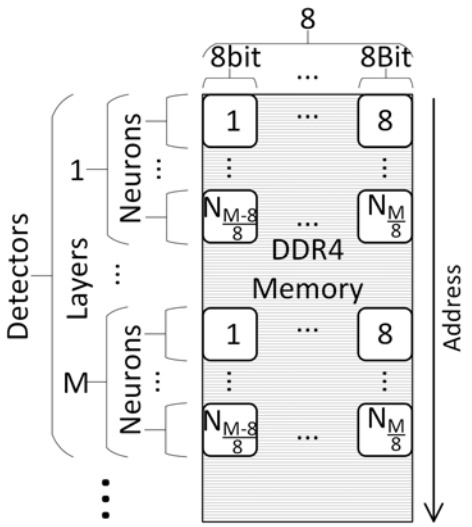
The weights are ordered in memory as shown in Fig. 3.4b. The memory bandwidth of 64 bits is divided into eight 8-bit streams. All weight streams are independent of each other. Each detector's weights are serialized into the memory with a linearly increasing address.

Huffman Decoding is a lossless coding scheme that takes the probability of encoded source symbols into account [80]. The more common a source symbol is, the shorter the assigned codes are. This is a good strategy as the typical weight distribution in our detector network follows a normal distribution. Most neural network accelerators for FPGAs do not consider weight compression via Huffman decoders. Han et al. propose an accelerator for deep compression according to [65]; however, in the accelerator design, Huffman decoding is not considered [81]. There is a research interest in using Huffman decoding for resource-limited devices [82]. More advanced compression methods such as the Deflate-algorithm use Huffman coding as part of the algorithm and high-speed hardware implementations have been proposed [83, 84]. However, we are unaware of any hardware implementations of these compression algorithms specifically for neural network accelerators. Our work implements Huffman coding as part of the deep compression approach from [65].

The eight bit-wide memory streams are chosen to simplify the Huffman decoder. If the bit-width of the input of the Huffman decoder is much larger than the average length of the codeword and a new input can be provided every clock cycle, then the Huffman decoder has to decode multiple codewords per clock cycle to match the throughput of the input. Since the Huffman decoder is



(a) Off-chip weight access architecture



(b) DDR4 memory layout for continuous address access

Figure 3.4: Memory layout and access structure for accessing off-chip weights.

a variable-length code, this would result in high circuit complexity. On the other hand, if the memory word is divided into too many independent streams, many independent Huffman decoders are needed. Each of these needs input and output FIFOs for compensating throughput fluctuations. As a result, the design would require a large number of FPGA resources. To strike this balance we chose eight Huffman decoders per 64-bit interface. The encoded data is organised as shown in fig.3.4b and each of the eight streams is encoded/decoded independently. The architecture of the proposed Huffman decoder is similar to the parallel decoder of [85], with the difference that up to two source symbols can be recovered per clock cycle. For the simplicity of the hardware, we chose to concatenate all layers per detector, and then Huffman code the resulting streams. This reduces the efficiency of the Huffman coding slightly as the per-decoder weight variance is larger than each single per-layer weight variance. In our trained and quantised network with the above-reported performance, we achieve a compression rate of 0.780 while the average per-layer compression rate is at 0.746. For 6-bit quantisation, this leads to an average length of 4.68 bits per weight.

It is important to match the Huffman decoder's throughput with the memory, and MAC throughput not to create throughput bottlenecks. Since each Huffman decoder is provided with eight bits at $\approx 92\%$ of all clock cycles and an average of 4.68 bits are required to encode one weight, the throughput of the Huffman decoder would be too low if it only decoded one weight per clock cycle. The proposed architecture can recover up to two source symbols per clock cycle and is detailed in Fig. 3.5. Each decoder has an input and output FIFO to decouple datarate fluctuations. A counter keeps track of the valid symbols in the buffer register. If the counter is too low, then eight new bits are written to the buffer register after the last valid data in the register. Two look-up-based decoders are available; both work fully in parallel and can decode up to one codeword per clock cycle. The second decoder is connected to 4 bits below the MSB of the buffer register. It speculates that the first decoder decodes a 4-bit codeword and only validates its results once this is confirmed. The probability of decoding a 4-bit codeword of any Huffman codebook is

$$P_{Dec4bitmatch} = \frac{\sum P_{all4bitcodes}}{\sum P_{allcodes}}. \tag{3.7}$$

With the fully trained weights of one of our detectors $P_{Dec4bitmatch} = 0.52$. The total average throughput thus is

$$\begin{aligned} Thr_{Huff} &= (P_{Dec1Match} + P_{Dec4bitmatch}) \times AvgCodelen \\ &= (1 + 0.5) \times 4.68 = 7.02\ [bits/clockcycle]. \end{aligned} \tag{3.8}$$

The goal of the Huffman decoder is to match its throughput Thr_{Huff} as closely as possible to the memory interface's throughput. The memory interface throughput is $8\ bit \times 0.92 = 7.36\ [bits/clockcycle]$. This is slightly larger than the Huffman decoder's throughput but is a close enough match for good pipeline performance.

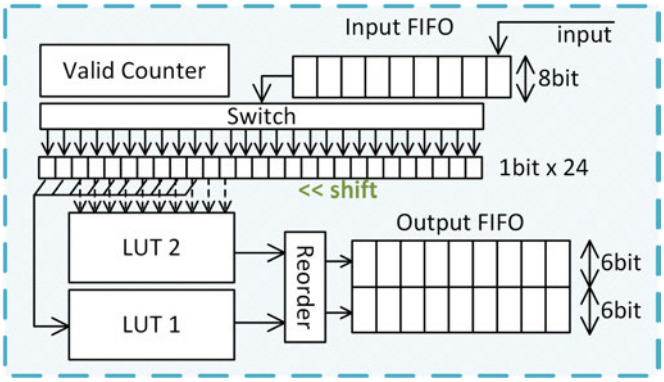


Figure 3.5: Superscalar Huffman decoder architecture with a speculative lookup of the second codeword

3.4.2 Small Batch Processing

As the design is memory-bound for off-chip weight access, the throughput of processed OFDM symbols can be increased by using any read weight on multiple OFDM symbols at once. This is termed batch processing and is used in certain accelerator designs for model inference [78, 86]. This work adds another application to the list where batch processing can be useful. The drawback of batch processing is of course higher detection latency. The

work by Posewsky and Ziener proposes a hardware implementation of a batch parallel FPGA accelerator [77]. The accelerator in this work is most noticeably different in that it uses not only batch parallelism, but also other parallelisms as laid out in the above loop unrolling strategy. Also, the Huffman coding, reordering of input symbols and multiple buffers for branching is demanded by the communication workload and distinguishes the proposed accelerator from [77]. The work in [78] focuses on batch processing in CNNs.

The batch processing starts at the symbol-input-buffer, which temporarily stores incoming symbols to be processed later in parallel as shown in Fig. 3.6. The input buffer accepts two input 16-bit words simultaneously and 128 input words are written per detector and per OFDM symbol consecutively. For the Dynamic Random-Access Memory (DRAM) accelerator, the input buffer is larger, as it can hold up to nine OFDM symbols for up to eight different detectors in memory. As the URAM accelerators are working on one neural network only, a much smaller input buffer is required, which can hold up to 2 symbols. All input buffers address the BRAM storage in such a way that previously stored OFDM symbols can be processed by the accelerator in parallel per detector. Writing and reading to the input buffer are independent, such that while data of one detector is read for processing, the data of another detector is written to the buffer to be processed next. As the Huffman decoder provides up to two weight values per clock cycle, the input buffer provides two input values per symbol too. The loop unrolling strategy also implies that the inputs have to be streamed to the processing unit multiple times, which is controlled by an input to the buffer.

3.4.3 Intermediate result buffer with branching support

The proposed accelerator processes each layer consecutively, i.e. no layer-level parallelism is supported. This is a common strategy but requires the buffering of one layer's results such that they can be used in the next layer. The layer outputs are small enough to be stored in an on-chip BRAM buffer - the so-called intermediate result buffer. It consists of two BRAM blocks that

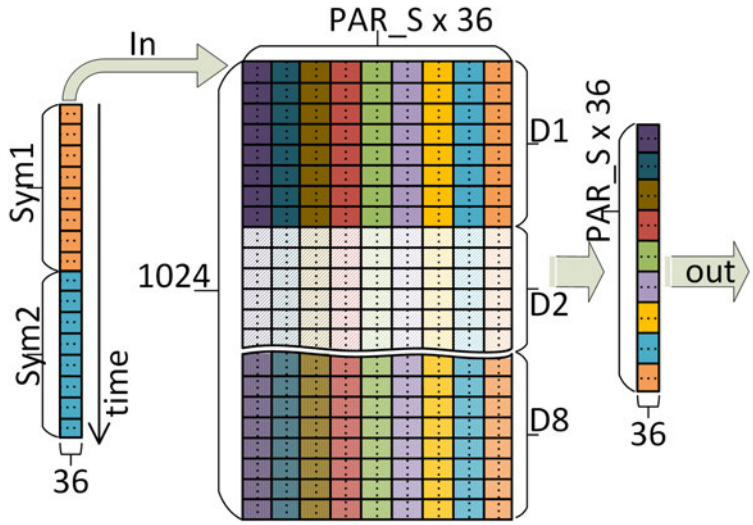


Figure 3.6: Input buffer optimised for small size batch processing

can support the size of up to 1024 Neurons for up to PAR_S parallel symbols. The Accelerator controller chooses which buffers to use in accordance with Table 3.1. The two buffers support branching in the computational graph of the neural network model without performance degradation. The result of the last common layer - layer two in this case - is stored in buffer 2. Buffer 2 is not overwritten until a new detector is calculated. The result in buffer 2 is used in all calculations of layer 3. If branching was not supported by the intermediate result buffer, costly off-chip memory writes would degrade the detector performance, as each layer would overwrite the layer 2 results.

Table 3.1: Buffer usage for the branching computational graph

Layer	1	2	3.1	4.1	..	3.8	4.8
Source Buffer	Input	B1	B2	B1	..	B2	B1
Target Buffer	B1	B2	B1	Out	..	B1	Out

3.5 Real-time Accelerator Design

By considering the above design considerations, a novel neural network accelerator specifically for the workload in communications is proposed next. The accelerator has two versions. One version holds the weight data in off-chip DDR4 memory banks. This version is ultimately limited by memory throughput. The second version holds the weights in on-chip URAM buffers. The available memory bandwidth from URAM to DSP blocks is large (up to 72 bits per clock cycle per URAM). The design is ultimately limited by the number of parallel URAM blocks per detector. Apart from the weight access, the biggest difference between the versions of the accelerator is the setting of PAR_N and PAR_S. The proposed accelerator architecture is depicted in Fig. 3.7.

As will be seen in Section 3.6.1, pruning does not provide a major performance benefit as for a reasonable pruning rate, the detector performance is degraded heavily. For this reason, and in order not to increase the hardware complexity further, pruning is not considered in the accelerator design.

3.5.1 Architecture for off-chip weights

In the DRAM accelerator $PAR\_N = 8$ and $PAR\_S = 9$. The input buffer provides access to collected symbols for the MAC unit. With $PAR\_S = 9$, 18 input-values of 9 different symbols are provided to the MAC unit per clock cycle. Similarly, the Huffman lookup tables provide 16 weights of 8 weight streams per clock cycle. As the weights are reused on each symbol, it results in 288 MAC calculations per clock cycle. Once the calculations for one set of output neurons are completed in the MAC unit, the 288 results are transferred to the writeback buffer in one clock cycle.

The Huffman decoder provides weight indices from 0 to 63 to the accelerator for the 6-bit quantisation, which strikes a good trade-off between accuracy and quantisation as shown in Section 3.6. These indices are used to look up an 18-bit fixed point weight value. Per Huffman decoder, one dual-port

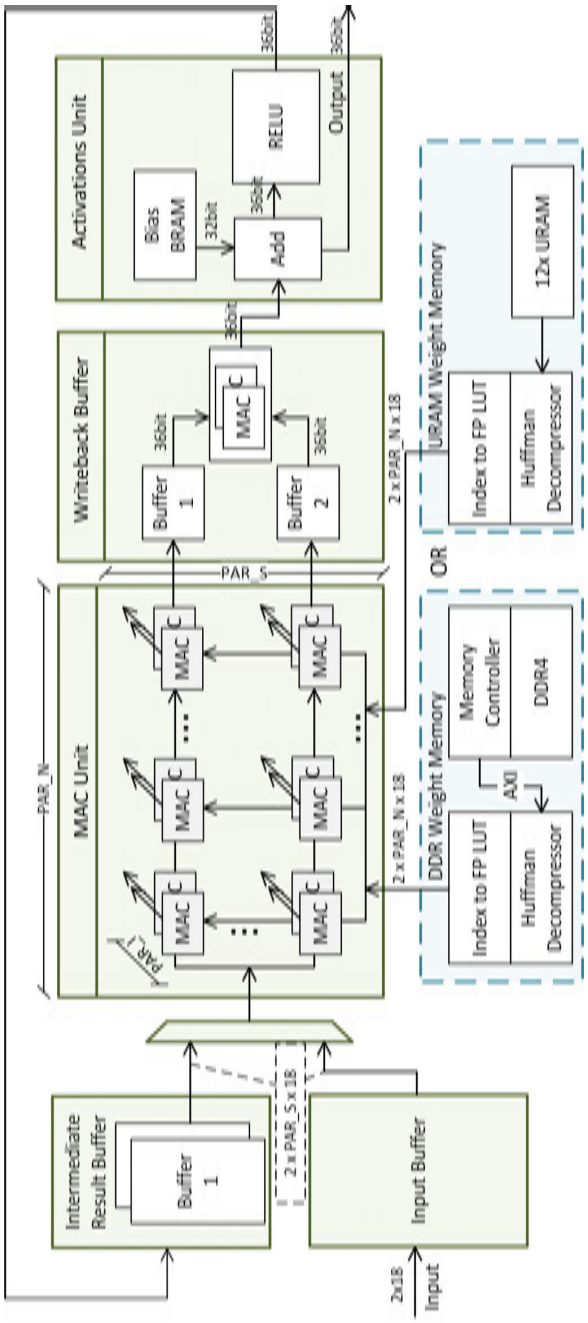


Figure 3.7: Proposed Neural Network Accelerator with Batch processing support, Huffman decoders and multiple intermediate result buffers for branching support

index translation Read Only Memory (ROM) is instantiated, which can provide separate lookup values for each layer of the accelerator. As each layer is quantised separately during the training phase, it is important for performance to also follow this quantisation strategy in hardware.

Once the MAC results have been written in the writeback buffer, the 128 values of the two buffers are added one per clock cycle to form 8 neuron values for 9 symbols. The writeback of data through the activation unit and into the intermediate result buffer can be executed in series as the neural network layers are large enough not to immediately require the last output data of the previous layer. A bias value is added to each result on the pass through the activation unit. This bias value is stored in a dual-port ROM as an 18-bit fixed point value. For each layer and each detector, different bias values are stored. All layers use the ReLU activation function, except the output layers. The ReLU function is easy to implement in hardware. The Sigmoid activation function of the output layer is not implemented in the accelerator, as its main function is to produce a predictive value between zero and one. It can be implemented in successive hardware or as a software function as the timing of it is not critical.

3.5.2 Architecture for on-chip weights

The URAM detector architecture is similar to the DRAM one in that it uses the same Huffman decoders, the same principle of intermediate result and writeback buffering, and the same activations unit. Apart from using on-chip URAM weight storage, one of the main differences is that $PAR\_N = 18$ and $PAR\_S = 2$. Because of the size limitation in the URAM, each accelerator only accelerates one detector network. As the processing latency is low, $PAR\_S = 2$ is sufficient to process the detector network in real time. For the best usage of the URAM resources, we chose the URAM memory data-with with 144 bits, utilising two parallel URAM blocks. The 144 bits translate into 18 Huffman

decoder modules, each receiving 8 bits of the data. $PAR\_N = 18$ follows, and a total of 36 MAC units are utilised. The writeback, activation, and intermediate result buffer are similar to the DRAM version of the accelerator except for adaptations to the differing parameters PAR_N and PAR_S.

3.6 Results and performance

This section reports the results of the previous design steps, from detection performance in terms of Bit Error Ratio (BER) to real-time capability and accelerator circuit metrics. Table 3.2 summarises the key parameters used to generate the presented results.

Table 3.2: Key Setup Parameters

Channel Model	WINNER II channel model as in [16]
SNR Range	5-25 dB
Pilot- to Data- Symbol Ratio	1/8
Number of training channel realisations	3 Million
Number of testing channel realisations	1 Million
Modulation Type	QPSK
Cyclic Prefix length	16 Samples
Number of OFDM SC per Detector	64
Number of Layers	4
Number of Neurons	566,400
FPGA Type	Xilinx xczu27dr-ffve1156-2-i
Total Memory Throughput	37.2 GB/s
Clock Frequency	300 MHz

3.6.1 Detection Performance

The training is performed across the range of SNR values from 5dB to 25 dB. Our experiments did not notice any significant performance improvement when training separately for each SNR value. For this reason, SNR values are picked randomly from the discrete set of SNR values in the training range, such that the receiver is trained for all SNR values in the range simultaneously. In Fig. 3.8a, the comparison with [16] shows that in our training, the performance of the same neural network structure leads to a 2 dB SNR loss at a BER of

10^{-2}. The discrepancy might have to do with extensive training times, as in [16] 20000 epochs are trained, whereas we stop at 2000 epochs for practicability reasons. However, the trained detector still outperforms traditional approaches by a large amount.

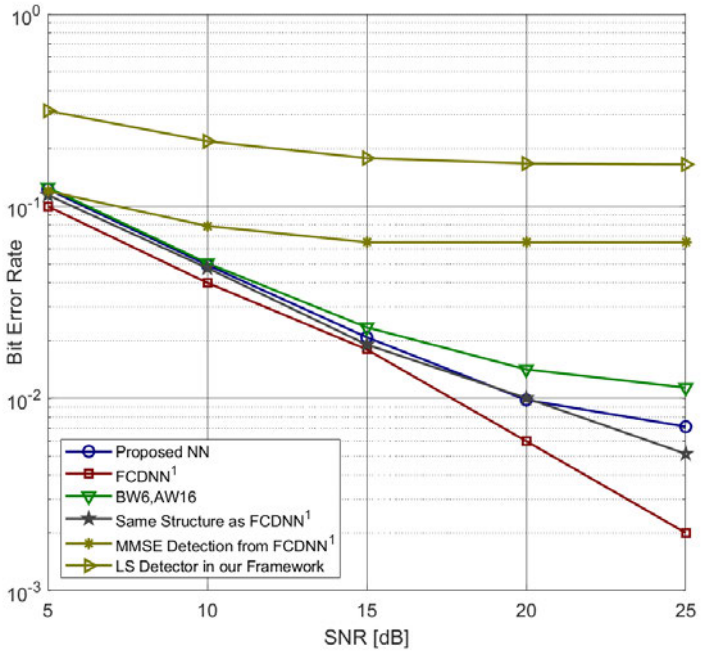
Another small performance degradation can be observed in Fig. 3.8a by comparing the proposed reduced complexity network with the neural network structure from [16] trained in our framework.

Quantisation with a weight bit-width of six bits and activation bit-width of 16 bits leads to another small performance degradation at high SNR values. We investigate the effects of quantisation on the network in depth and define the accuracy-loss with respect to the floating-point BER for each configuration as

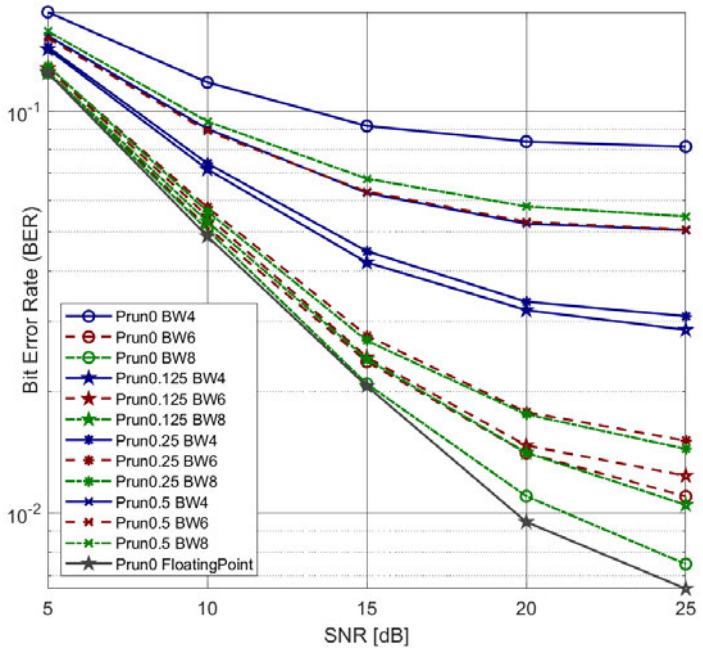
$$QErr = \frac{BER_{Quant}}{BER_{FP}} - 1. \tag{3.9}$$

In Table 3.3a, the $QErr$ when the detector is trained in a SNR range from 5 to 25 dB is provided. In Table 3.3b, the $QErr$ at 25dB is shown. A higher impact of quantisation in higher SNR values can be observed.

Fig. 3.8b shows the detection performance when pruning is introduced on top of quantisation. The pruning rates of 12.5%, 25%, and 50% are compared to the unpruned, quantised, and floating point networks. For pruned networks, quantisation is started after 500 epochs and pruning after 750 epochs. The total number of trained epochs is 2000. For 8-bit weight quantisation, pruning of 12.5% leads to a performance loss of ≈ 5 dB at 25 dB. This is similar to the performance loss when quantising weights with 6 bits. The network with 6-bit quantised weights suffers from performance loss of ≈ 2.5 dB and ≈ 6 dB by pruning it to 12.5% and 25%, respectively. The memory requirement for the configuration of Prun0.125BW8 is $\approx 16\%$ higher than for Prun0BW6 as seen in Fig. 3.9b. Because of this and the fact that pruning adds additional hardware complexity, pruning is not considered in the accelerator design.



(a) Detection performance comparison for various neural networks for the range of 0dB to 25 dB. [1]FCDNN [16].



(b) Detection performance loss due to simultaneous pruning and quantisation.

Figure 3.8: Performance for quantised and pruned neural network detectors.

			Bitwidth Weights [bits]				
			4	5	6	7	8
Quantization Aware Training	Bitwidth Activations [bits]	6	657.0%	422.0%	388.0%	373.8%	67.1%
		8	263.5%	98.8%	67.1%	58.7%	59.8%
		10	186.4%	46.3%	18.7%	14.7%	12.9%
		16	163.0%	33.6%	7.2%	1.7%	2.4%
Post Training Quantization	Bitwidth Activations [bits]	6	768.3%	489.1%	398.1%	368.7%	487.0%
		8	479.5%	192.0%	99.7%	75.7%	71.4%
		10	390.1%	121.8%	38.5%	20.0%	16.2%
		16	345.3%	73.5%	9.7%	6.3%	0.3%

(a) Average performance degradation $QErr$ from Eq. (3.9) in percent, when training the detector with SNR values from 5 to 25dB.

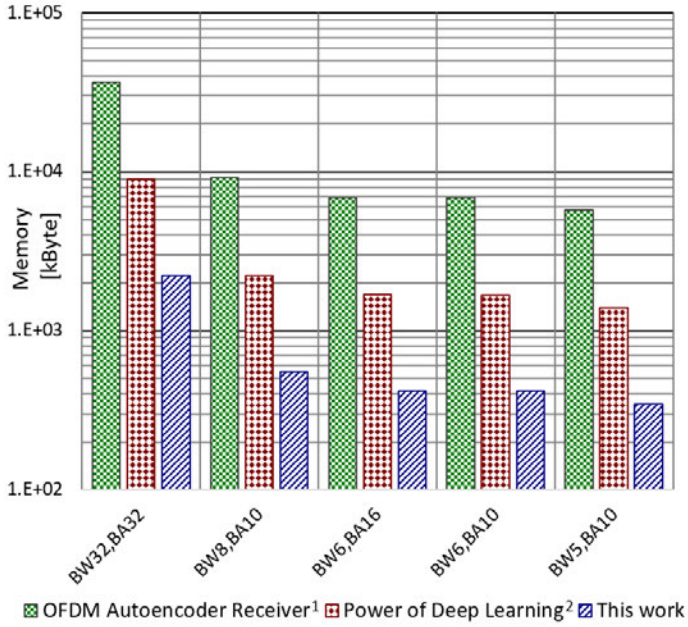
			Bitwidth Weights [bits]				
			4	5	6	7	8
Quantization Aware Training	Bitwidth Activations [bits]	6	4057%	2383%	2153%	2001%	297%
		8	1610%	505%	297%	226%	223%
		10	1167%	259%	85%	68%	52%
		16	1044%	193%	54%	10%	5%
Post Training Quantization	Bitwidth Activations [bits]	6	4749%	3111%	2111%	2085%	2740%
		8	3052%	1131%	524%	318%	306%
		10	2516%	751%	229%	85%	66%
		16	2277%	558%	112%	19%	-8%

(b) Average performance degradation $QErr$ from Eq. (3.9) in percent, when training the detector at 25dB SNR.

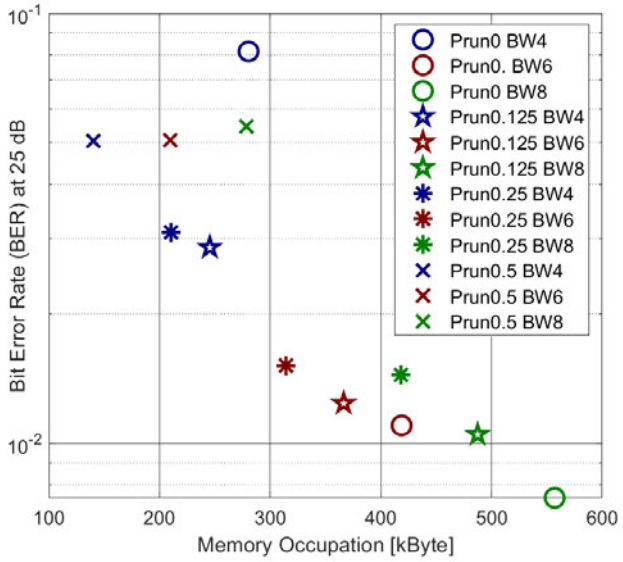
Table 3.3: Impact of quantisation on performance.

3.6.2 Memory Requirement

As a baseline for memory-requirement comparison, single-precision floating-point parameters with 32 bits per parameter according to [87] are assumed. For comparison, the autoencoder from [22] is included. Each sub-carrier requires one autoencoder in this design, making it the most complex design. For fairness, only the receiver network of the autoencoder is considered. Fig. 3.9a shows significant savings from reduced complexity models and quantisation. The novel model results in a memory reduction of ≈ 4.02 times, and the quantisation with BW=6 and BA=16 results in another memory reduction of ≈ 5.3 times as compared to 32-bit floating-point weights. Huffman decoding further reduces the memory requirement by $\frac{1}{0.780} \approx 1.28$, leading to a total memory reduction of $4.02 \times 5.30 \times 1.28 \approx 27.27$ times.



(a) Memory requirement to process 64 sub-carriers compared to previous work in the literature. [1][22] [2][16].



(b) Memory requirement and detection performance of various quantisation and pruning options. All activation weights are assumed to be 16 bit except floating point activations are 32 bits.

Figure 3.9: Memory requirement to process 64 SC for various combinations of quantisation and pruning settings.

3.6.3 Scheduling and Latency

The performance of the presented accelerator is evaluated on the Xilinx Zynq Ultrascale+ XCZU27DR-2 FPGA for up to 1200 SC. We evaluate the performance for multiple numbers of SC commonly used within standardised LTE numerology [19]. The largest configuration of 1200 SC provides a 20 MHz channel bandwidth at a sub-carrier spacing of 15 kHz. One of the main performance metrics of the accelerator is the detection latency in symbol durations. The measurement in symbol durations is useful as it makes the latency independent of the underlying OFDM numerology. With 15 kHz, the symbol duration is $T_S = \frac{1}{15kHz} \approx 66.7\mu s$.

As described in Section 3.5.1 and Section 3.5.2, the accelerators allow for buffering and simultaneous processing of multiple symbols. While one detector is processed, all detectors including the currently processed one, buffer the newly received symbols. This batch processing has an impact on system latency. The number of detectors is the main factor for the maximum latency besides the processing time

$$L_{DDR\_max} = \lceil N_{DDR\_det} \times T_{DDR\_proc} \rceil + T_{DDR\_proc}. \tag{3.10}$$

N_{DDR_det} is the number of parallel detectors per accelerator. T_{DDR_proc} is the processing time for each detector. The ceiling function indicates that processing cannot start while a symbol is not completely received yet. The average detection latency for each of the DRAM accelerators is

$$L_{DDR\_avg} = \frac{\sum_{k=1}^{n} k}{n} + T_{DDR\_proc} \tag{3.11}$$

$$= \frac{n+1}{2} + T_{DDR\_proc} \; ; with \; n = \lceil N_{DDR\_det} \times T_{DDR\_proc} \rceil. \tag{3.12}$$

Leading to a combined detection latency of:

$$L_{avg} = \frac{N_{DDR\_det} \times L_{DDR\_avg}}{N_{det}} + \frac{N_{URAM\_det} \times L_{URAM\_avg}}{N_{det}}. \tag{3.13}$$

As each URAM accelerator processes only one detector ($N_{det} = 1$), L_{URAM_avg} is equal to T_{URAM_proc}. Because each accelerator processes one or more detectors which are assigned during design time, each detector's throughput has to be

$$TP_k = \frac{PAR\_S_k}{T^k_{proc} \times N^k_{det}} > 1. \tag{3.14}$$

Where k is the detector index. If the throughput falls below the value of 1, symbols will build up at the input, and the processing latency will continuously increase. Since the physical input buffers have limited capacity, it will eventually result in data loss, violating the real-time condition. As the assignment of detectors to accelerators is fixed, a build-up of symbols at one or more detectors might even happen if the total system throughput satisfies

$$TP = \frac{1}{N_{acc}} \sum_{k=1}^{N_{acc}} TP_k > 1. \tag{3.15}$$

Therefore, Eq. (3.15) is not a sufficient condition for guaranteeing real-time processing. Table 3.4 shows the accelerator allocation for maximum performance for different numbers of processed SC. First URAM accelerators are allocated to maximise performance. The latency is given according to Eqs. (3.10) to (3.13). The throughput per detector and the total throughput in terms of symbols according to Eqs. (3.14) and (3.15) are shown respectively. Up to 13 detectors can be deployed while the real-time condition still holds. The average throughput per detector is 1.4 symbols per T_S and all symbols can be processed in time. A maximum symbol latency of 11.52 symbols, and an average latency of 3.67 symbols results. The 13 detectors amount to a total of 832 QPSK modulated SC allowing a bitrate of $832 \times 15kHz \times 2 = 24.96\frac{Mbits}{s}$ including pilot symbols. For 900 SC, the throughput in the DRAM accelerators is

too small in our configuration, even though the average throughput per detector would be sufficient as indicated by the colouring of the last row in Table 3.4. For 1200 SC, each of the DRAM accelerator's throughput and average throughput is too low as indicated by the red colouring.

Table 3.4: Accelerator performance in terms of processing latency, throughput and real-time capability for the various number of SC. Latency is calculated according to Eqs. (3.10), (3.11) and (3.13) and throughput according to Eqs. (3.14) and (3.15). Light green fields indicate sufficiently large throughput for real-time processing according to Eq. (3.14).

Bandwidth[MHz]	1.4	3	5	10	11.52	15	20
Number of SC	72	180	300	600	832	900	1200
Required Accelerators (Number of SC / 64)	1.125	2.8125	4.6875	9.375	13	14.0625	18.75
URAM Accelerator Processing Latency [Sym]	1.087						
URAM Accelerator Throughput [Sym/TS]	1.840						
DRAM Accelerator Processing Latency [Sym]	2.519						
Allocated URAM accelerators	2	3	5	6	6	6	6
Allocated DRAM accelerators	0	0	0	2	2	2	2
Detectors per DRAM accelerator1/accelerator2	0/0	0/0	0/0	3/1	3/3	5/4	7/6
Maximum DRAM accelerator latency [Sym]	-	-	-	11.52	11.52	17.52	23.52
DRAM accelerator 1 throughput [Sym/TS]	-	-	-	1.19	1.19	0.71	0.51
DRAM accelerator 2 throughput [Sym/TS]	-	-	-	3.57	1.19	0.89	0.60
Average DRAM accelerator latency [Sym]	-	-	-	4.76	6.26	8.51	11.51
Average symbol latency [Sym]	1.09	1.09	1.09	2.56	3.67	5.54	8.22
Maximum throughput [Sym/TS]	4.35	6.52	10.87	18.19	18.19	18.19	18.19
Average throughput per detector [Sym/TS]	2.17	2.17	2.17	1.82	1.40	1.21	0.96

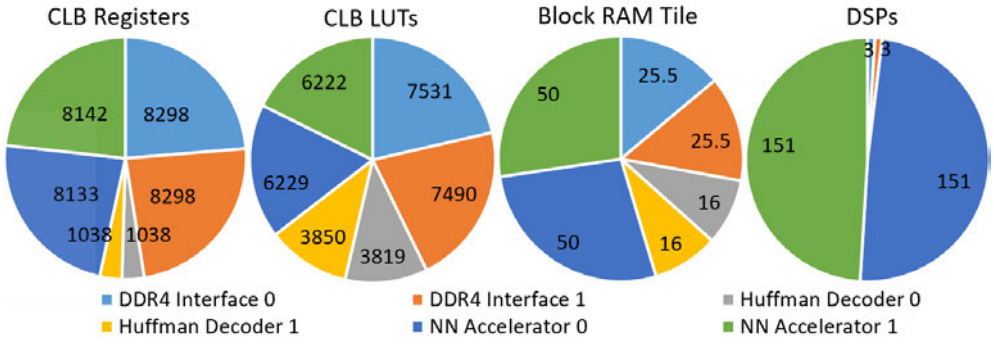
3.6.4 Circuit Metrics

Each DRAM accelerator is clocked at 300 MHz. This clock is provided by the physical DDR interface IP and is dependent on the connected memory chips. The memory chip is clocked at 1200 MHz and transfers 8 bits of data per positive and negative clock edge. This translates into a bus with 64 bits data-width at a transfer rate of 300 MHz (single clock edge). The URAM accelerator's maximum frequency is 300 MHz as well. The critical paths are manifold, with the most noticeable ones being in the accelerator's control logic and in the flexible shift mechanic of the Huffman decoder's buffer register.

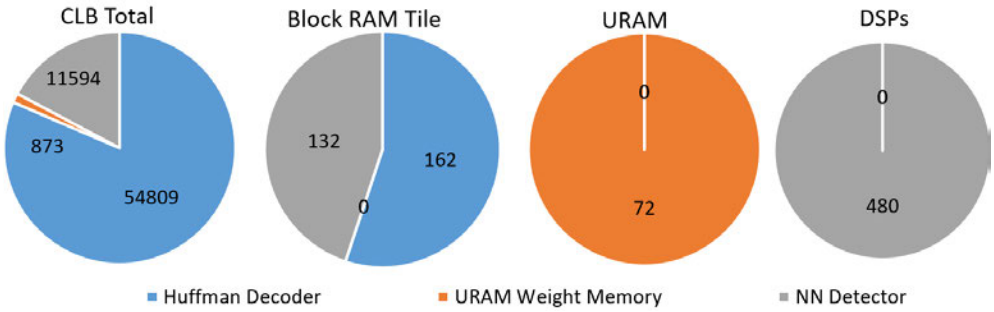
Table 3.5 shows the average resource usage per single accelerator, while Fig. 3.10 shows the average allocation of resources per submodule per accelerator. We notice that for the DRAM accelerator, more than half of all Configurable Logic Blocks (CLB) resources and slightly less than half the BRAM resources are used by the physical memory interfaces. The DRAM accelerator uses more resources, but can also process multiple detectors.

Table 3.5: Average FPGA resource utilisation for each accelerator version. A non-integer number of resources result from averaging the resources over multiple accelerators.

	CLB LUTs	CLB Registers	Block RAM	URAM Blocks	DSPs
DRAM Accelerator	17.5k	17.5k	91.5	0.00	154
URAM Accelerator	8k	3.7k	49	12.00	80.7



(a) Resource utilization for two DRAM accelerators



(b) Resource utilization for six URAM accelerators

Figure 3.10: Resource usage per submodule per type of accelerator for two DRAM accelerators and six URAM accelerators

3.6.5 Accelerator Efficiency Evaluation

To assess the implementation efficiency of the proposed accelerator, we compare the measured performance in terms of operations per second with its theoretical maximum performance. This theoretical maximum performance assumes that all computational units perform useful computations every clock cycle and that the available memory bandwidth transports data continuously at its full capacity. The measure of operations per second is adequate to assess performance as it directly relates to the throughput of an accelerator for a given neural network. Table 3.6 lays out the maximum achievable performance given the DSP utilisation and the available memory bandwidth. It can be seen that both accelerators are memory-bound, as expected, for fully connected network accelerators. The measured detector performance was obtained by digital simulation and was found to be 9.6% and 15.3% lower than the theoretical maximum performance of the DRAM and URAM accelerator, respectively. This inefficiency is mostly due to the varying length of the weight streams after Huffman coding. Some streams can have regions where the decoders can only decode one weight per clock which leads to a stall of the processing elements. However, The benefit of the Huffman compression still outweighs this inefficiency.

Table 3.7 compares our work with previously proposed neural network accelerators. It is important to notice that the latency and throughput figures have been extracted for fully connected layers from the previous work for a fair comparison. The utilisation figures have been extracted where possible. From the comparison, it can be noticed that even though our DRAM accelerator has much smaller memory bandwidth available, its throughput is considerably larger than the throughput in [88] and [89]. This can be attributed to a few factors: the weight quantisation to 6 bits, the Huffman compression, and the batch processing, to which we attribute the highest performance gain. In addition, also the older technology nodes of 28 nm and the lower operating frequencies in [88], and [89] are contributing factors. The design in [90] uses batch processing for its FC layers and uses a batch size of 128. We estimate

Table 3.6: Theoretical and measured accelerator performance

Accelerator Performance	DRAM	URAM
Measured Detector Performance without Activations		
Number of weights in detector network	566400	566400
Processed symbols per second [Sym/s]	15000	15000
Accelerator throughput [Sym/Ts]	1.19	1.84
Detectors per accelerator	3	1
Performance [GOP/s]	30.36	15.64
Theoretical Maximum Performance based on DSP Utilization		
Accelerator frequency	3E+08	3E+08
Number of DSPs per Accelerator	154	80.67
Theoretical maximum performance based on the number of DSP [GOP/s]	46.2	24.20
Theoretical Maximum Performance based on Memory Bandwidth		
Buswidth [Bytes]	8	18
Busspeed [MHz]	300	300
Memory throughput [MB/s]	2400	5400
AXI-Interface throughput reduction factor	0.91	-
Average number of bits per weight [bits]	4.68	4.68
Average number of weights processed per second [MWeights/s]	3733.33	9230.8
Maximum batch size	9	2
Theoretical maximum performance based on memory bandwidth [GOP/s]	33.60	18.46

the latency for the Alexnet calculation in [90] to be $125.5\,\mathrm{ms}$ as compared to the latency for the same neural network in [88] of $12.75\,\mathrm{ms}$. This shows how a large batch size like this trades off throughput for latency. The proposed design achieves ≈ 9 times lower throughput than [90] with ≈ 2 times the total memory bandwidth. However, its batch size is ≈ 14 times and 64 times smaller for the DRAM- and URAM- accelerator, respectively.

Table 3.7: Accelerator performance comparison

	[88]	[89]	[90]	This Work	
				One DRAM Accelerator	All Accelerators
Model	Alexnet	VGG16-SVD	Alexnet	OFDM Detector	
Quantization	fixed 8 bit	dynamic 16 bit	fixed 16 bit	fixed 6 bit	
FPGA	Stratix-V GXA7	Zynq XC7Z045	Arria 10	RFSoC (Ultrascale+)	
Feature Node	28nm	28nm	20nm	16nm	
Clock Frequency[MHz]	100	150	300	300	
External Memory	2 DDR3 x64	1 DDR3 x64	1 DDR4 x64	1 DDR4 x8	2 DDR4 x8 + 6 URAM x18
Throughput					
Memory Bandwidth [GB/s][a]	12.8-29.87[b]	4.2	17	2.40	37.2
Weights in the FC Layers	59M	73M	59M	566k	7.4M
Average Latency FC Layers [ms]	2.83	61.18	125.5[c]	0.0187	0.24
FC Layer Throughput [GOP/s]	20.72	1.20	1382.00	30.33	154.56
Utilization					
DSP Utilization	256	780	1476	154	788
Logic Utilization[d]	15.48K[e]	310.27K	246K	35.04K	139.88K
BRAM/URAM Utilization	213/-	486/-	2487/-	91.5/-	477/72

a. 1GB ≙ 1e9B
b. Not clearly specified, estimation based on DDR3 specification
c. Processes the FC layers in batches of 128 after calculating all preceding layers first. As the FC layers are at the end of the network we can get a rough estimate based on the average image latency ($\frac{1}{1020\ \mathrm{img/s}} = 0.98\ \mathrm{ms/img}$) multiplied by the batch size of 128.
d. Xilinx FPGAs in Total CLBs (Registers + LUTs) and Altera FPGAs in ALMs
e. Resources directly associated to FC6&FC7&FC8 Layers

3.6.6 Comparison with Traditional Approaches

Computational Resources

The neural network based detector outperforms traditional approaches significantly, as seen in Fig. 3.8a. This gain in performance comes at the expense of computational complexity. For example, a modified MMSE FPGA implementation is reported in [91]. Timing is not reported in detail, but resource utilisation of LUTs, Registers, and DSP blocks is very low. The memory requirement with $\approx$

150 kB is within the same order of magnitude as what is needed in our design to process 64 SC. The accelerator in [91] however, processes 840 SC. The work in [92] presents a linear MMSE detector that shows reduced complexity by low-rank approximation via singular value decomposition. Assuming a rank of 6, and 8 pilots per 64 SC, the number of complex multiplications for estimating the channel transfer function is reported as 105. This is much lower than the proposed neural network's required number of operations. The channel estimator and OFDM equaliser provided in Mathworks' Wireless Hardware Description Language (HDL) toolbox implements basic least square channel estimation and zero-forcing or minimum mean square error equalisation [93, 94]. Both designs achieve a clock frequency of 244.6 MHz on the Xilinx Zynq- 7000 ZC706 evaluation board. Their hardware resource requirements are shown in Table 3.8. The resource usage in terms of LUTs is comparable to the URAM accelerator, while DSP and memory resources are far lower.

Table 3.8: Resource usage in Mathworks' OFDM channel estimator and equalizer

Resource	Channel Estimator [93]	Equalizer [94]	Total [93] + [94]	URAM Accelerator
Slice LUTs	2684	7380	10064	7976
Slice Registers	1184	8063	9247	3656
DSPs	6	24	30	80
BRAMs	1.5	0	1.5	49
URAMs	-	-	-	12

Latency

The LTE-A user-plane latency is typically specified to be less than 10ms [19, Chapter 27]. However, the processing time is specified by 3GPP as 1.5ms for UEs [95]. Comparing this time with the maximum latency for the DRAM accelerator of 11.52 symbols ($\hat{=}\ 0.77\,\mathrm{ms}$) and the URAM accelerator of 2.52 symbols ($\hat{=}\ 0.17\,\mathrm{ms}$), this seems possible. However, a typical OFDM receiver consists of more than just channel estimation and detection circuits. The latency of the traditional approaches in [93, 94] depends on the detailed setting of the block

but is within the range of a few clock cycles up to 951 clock cycles for more advanced settings. This shows, that in comparison to the neural network based accelerator, there is a large gap in latency. The specific design and use case will determine whether or not this latency is acceptable.

3.7 Limitations and Future Research

This work investigates a generic OFDM system for channel estimation and signal detection. It represents the current state of the art in the field; however, more research is still required for deployment in an actual receiver. A gap still exists between the presented, generic OFDM system and 3GPP standards. We acknowledge the following points as the main limitations towards adoption and 3GPP standard compliance:

- The position of reference signals in the two-dimensional resource element lattice in time and frequency cannot be chosen dynamically as required by the 3GPP standard for LTE [96]. Instead, at every time instance, the decoder assumes the same interleaving of reference- and data- symbols in frequency (i.e. a comb-type OFDM system). Further, reference symbols are not chosen according to the 3GPP standard.
- The modulation scheme is fixed to QPSK, as the neural network is limited to that. In the current state of research, it is unclear how to efficiently extend neural networks' training and inference process to higher modulation schemes such as QAM64 or higher. We performed some experiments with multi-level activation functions to facilitate higher modulation orders; however, these experiments have not been successful yet. Another option is to perform detection at the bit level rather than at the symbol level. An increased number of neurons in the output is the consequence.
- The computational complexity has been reduced significantly in this work. However, the data-driven design methodology makes real-time processing still challenging. Model-driven designs might help with this. From an FPGA perspective, the trend to highly integrated HBM and larger on-chip memories can help to mitigate the current memory limitations.

Future work might focus on the next stages of the design process. The described IP can be integrated into a classical receiver processing chain and deployed without major restructuring of the receiver. Over-the-air measurements could provide valuable insights into retraining requirements and the stability and reliability of the neural network based approach. At the current stage, it is not clear how much retraining slowly changing channel statistics might necessitate. This potential retraining can be performed in the background based on the received data. Then, the receiver weights might be dynamically updated. This distinguishes the neural network receiver significantly from classical receivers. The architecture, cost, and feasibility of such an approach have to be investigated further.

3.8 Summary

In this chapter, we proposed one of the first neural network detectors for an OFDM communication system with real-time processing capability. The proposed detector utilises a simplified structure of a previously reported neural network performing efficient channel estimation and detection. Three main deep compression techniques were explored: pruning, quantisation, and weight encoding. The network complexity could be reduced this way; however, reduced performance was observed when pruning and quantisation are applied simultaneously. As memory throughput and memory size are the limiting factors, our efforts focus on reducing the memory requirement and achieved a reduction by ≈ 27 times. A neural network accelerator was developed to accelerate the targeted fully connected neural network. The accelerator has three features making it uniquely suitable for processing the proposed neural network in real-time. It allows for processing multiple symbols simultaneously in a batch. Loaded weights are used on multiple symbols, and throughput is increased. Multiple intermediate result buffers allow for efficient branching in the neural network. High-performance Huffman decoders with a decoding rate larger than 1 allow efficient usage of weight buffering FIFOs. The design can be expanded

by instantiating more accelerators; however, in our platform containing the Xilinx RFSoC, we simulated real-time processing for 832 sub-carriers with an average detection latency of ≈ 3.7 OFDM symbols. The thereby enabled datarate, including pilots, is just under 25 Mbit/s.

We consider the presented work an important step towards deployable, neural network detector solutions in communication systems. Even though the application of compression techniques reduced the memory requirement significantly, the computational complexity and memory requirements are still far beyond that of traditional approaches. The improved performance can justify this much larger complexity to some extent; however, we also recognise the need for further research in the area.

In the following chapter, a step away from a data-driven design towards a model-driven design is taken, in the hope of reaching a more scalable detector solution. Also, the second core detector technology in 5G-NR is investigated, namely massive MIMO. The detectors will be most suitable for base station deployment as the detection task for massive MIMO is located there. Also, the targeted Xilinx RFSoC is well-suited for base station deployment.

Chapter 4

Machine Learning enhanced, Real-time Massive MIMO Signal Detection

4.1 Introduction

In recent years, much research has developed around machine learning enhanced signal detection in massive MIMO. As explained in Section 1.1, the benefits include high detection performance by learning unknown or difficult channel conditions, by allowing for learned hardware impairment compensation, and by optimising across the classical block boundaries.

This chapter starts with a detailed review of proposed ML enhancements for massive MIMO in Section 4.2. Data-Diven designs are briefly reviewed, however; their suitability for real-time processing is limited due to the high computational complexity. A natural choice for model-driven ML-detectors are PGD based designs due to their intuitive, iterative solution to the detection problem. Presumably, for that reason, PGD detectors have been one of the first to be extended by ML-parameters in the literature. Another closely related and much-researched class of ML detectors are based on OAMP. They have advantages over PGD, such as faster convergence and are reviewed next. Other notable detector types are briefly discussed before reviewing the channel models used in the literature. The channel models are of decisive importance for estimating the detection performance of the algorithms. Finally, the limited literature on real-time processing of ML-enhanced massive MIMO detection is reviewed.

Since OAMP has been identified as a well-suited candidate, commonalities of the OAMP-Net [31] and MMNet [26] algorithms are identified, and a unified algorithm is proposed in Section 4.3. The proposed algorithm allows the computation of MMNet, i.i.d MMNet, OAMP-Net, and classical OAMP without changing the algorithm structure. Merely the inputs to the algorithm are assigned different values depending on the type. The algorithm's computational complexity is briefly reviewed to provide a first estimate of the accelerator complexity.

Next, the resulting algorithm is transformed into hardware-friendly operations, and a microarchitecture is defined in Section 4.4. A digital circuit is developed via Vitis-High Level Synthesis (HLS) and synthesised for the Xilinx RFSoC.

The accelerator's profiling results are presented for many system configurations and modulation types in Section 4.5. The pipelined latency of the detector is as low as $2\,\mu s$; however, the overall performance in terms of throughput and latency is lower compared to traditional (non-ML) accelerators in the literature. This accelerator serves as a baseline for Chapter 5, in which further improvements on the algorithm and hardware are proposed.

4.1.1 Contribution

The core contribution of this chapter is the unified algorithm allowing the implementation of MMNet and OAMP-Net in a common accelerator, its hardware architecture and implementation, and the profiling of the resulting accelerator. The resulting accelerator could replace a classical massive MIMO detector in a classical receiver processing chain as seen in Fig. 2.5. There are no particular requirements for this placement except for a way to provide the learned parameters to the accelerator.

The reported algorithm, accelerator architecture and digital circuit are covered by the publication in [28]:

S. Brennsteiner, T. Arslan, J. S. Thompson, and A. McCormick, 'A machine learning enhanced approximate message passing massive MIMO accelerator', in 2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS), Jun. 2022, pp. 443–446.

4.2 State of the Art

The extensive research interest in ML-enhanced massive MIMO detection resulted in many proposals based on various underlying algorithms. This section briefly reviews and classifies the most influential research based on the underlying algorithm type. Although this work focuses on massive MIMO, some MIMO proposals with a smaller number of BS antennas are discussed where appropriate. In particular, for real-time processing, little work has been done in the space of ML-enhanced massive MIMO detection.

4.2.1 Data-Driven Designs

The detection algorithms can be categorised based on whether or not domain knowledge is incorporated into their architecture. If no such knowledge is used, the algorithm is categorised as data-driven. Of course, model-driven algorithms also require data to learn from the experience; however, they are based on known algorithms. Data-driven designs typically consist of the classical layers used in deep learning such as FC-, CNN-, and RNN- layers as introduced in Section 2.4.

Autoencoder

One of the earliest proposals for a data-driven, machine learning-based signal detection in communications was made by the authors in [53, 97]. The transmitter, as well as the receiver, are built from multiple FC layers. The channel between the transmitter and receiver is modelled and implemented within the same training framework. In deep learning, the structure where a

learnable neural network performs a transformation to an intermediate signal representation and then another neural network performs the transformation to a target representation is referred to as **autoencoder**. Autoencoders are typically used for dimensionality reduction to improve performance on tasks such as classification [51]. In mobile communications, on the other hand, the goal is to transmit a certain amount of data over the channel and reconstruct it with the least number of errors on the receiver side. The work by O'Shea et. al. in [53, 97] has shown that the autoencoder can learn and optimise the used symbol alphabet during training, thereby improving detection performance. However, the tested MIMO configuration is just two receive- and two transmit- antennas. An autoencoder for hybrid beamforming in a massive MIMO downlink scenario is implemented and profiled in [98]. The autoencoder design philosophy might be the closest to a true neural network-based, end-to-end communication system as envisioned by some researchers [13]; however, it is still in its early development phase, and many open questions remain.

Deep-Learning Receivers

Another option is to rely on conventional symbol alphabets and only implement or enhance the receiver process with deep-learning-based structures. The neural networks **DenseNet** and **MobileNetV2** are originally designed for image classification tasks but can be repurposed for the MIMO detection problem [99]. The authors use these neural networks to replace multiple classical receiver blocks such as channel estimation, space-time coding, signal detection, and channel coding. This has the advantage of allowing a global optimisation process in the receiver. Results show performance gains in terms of reduced SER when using non-ideal channels as compared to a traditional detection algorithm. The authors test various MIMO system configurations of up to four receive and transmit antennas with BPSK and QPSK modulation. The computational cost of the receiver is, however, very high. For DenseNet, the number of multiplications is ≈ 312 k $\times (M_R \times M_T)$ while for MobilNetV2 it is ≈ 255 k $\times (M_R \times M_T)$. This large computational cost might be acceptable

for image processing where throughput and latency requirements are relatively low; however, in communications where requirements are high, it is currently infeasible to compute these neural networks in real time and at scale. Further, the technique does not scale well to massive MIMO.

Instead of re-purposing existing neural networks from other domains, the authors in [100] develop new neural networks from basic deep learning building blocks. Based on CNN-, RNN-, and FC- layers three neural networks are evaluated for a 2x2 MIMO system. A Rayleigh channel is used and the networks show close to maximum likelihood performance, however, it is not clear how varying channels are handled as the networks do not consider the channel matrix as an input. Further, the small number of antennas makes the results not indicative of actual massive MIMO performance and cost.

The work in [101] compares a CNN and DNN implementation of a detector for a 2x2 MIMO system. The authors in this work provide the channel matrix $\boldsymbol{H}$ as well as the received vector $\boldsymbol{y}$ as inputs to their detector and consider the impact of imperfect channel knowledge. The results show that these networks are capable of outperforming the traditional Zero-Forcing and MMSE algorithms as well as DetNet (see Section 4.2.2). The disadvantage of these networks is the immense computational complexity. Approximately 215k and 208k parameters are required for the DNN and CNN network, respectively. Considering the small MIMO configuration and the Modulation types of BPSK and QPSK, the scalability issue of the approach is clear.

The authors in [102] propose a partial learning method to efficiently tradeoff complexity and performance. First, a neural work consisting of multiple FC layers provides an estimate of the sent symbol vector $\widehat{\boldsymbol{x}}$, and then a zero-forcing detector with a decision feedback equaliser is deployed to arrive at the final estimation of $\widehat{\boldsymbol{x}}$. The work shows improvements over traditional methods in terms of SER for MIMO combinations of 8x8, 32x32, and 64x64 antennas with BPSK modulation. The neural network introduces $\approx$ 49k multiplications per de-

tection for the 64x64 case on top of the existing linear detection method, which is much less than other approaches. On the other hand, the used channel model is unrealistic for a mobile communication scenario, and the comparison to other ML-based algorithms is limited.

Data-Driven Design Limitations

At the current time, data-driven designs show limitations in their practical deployment. One of the main limiting factors is computational complexity. As seen above, detectors can have up to a few hundred thousand model parameters, even for a small number of receive and transmit antennas. As latency and throughput are primary performance metrics in mobile communications, running inference on these networks in real-time becomes infeasible.

Another concern with purely data-driven designs is that the domain knowledge acquired in decades of research and development is discarded. For many problems in communication science, good models and approximations of systems exist; for some problems, even analytical solutions can be provided. The purely data-driven design approach discards all that knowledge. It can result in overly complex solutions and a lack of robustness to changes in the underlying statistical properties of a system.

Pure data-driven solutions are intriguing from a conceptual point, especially the end-to-end learning solutions; however, their limitations are yet to be overcome effectively. Much recent research has therefore focused on a model-driven approach. The following section reviews some of the most influential model-driven designs recently proposed in the literature and elaborates on their strengths and weaknesses.

4.2.2 System Model and Projected Gradient Descent (PGD)

The complex-valued massive MIMO system can be described as introduced in Eq. (2.12)

$$\overline{\boldsymbol{y}} = \bar{\boldsymbol{H}}\overline{\boldsymbol{x}} + \overline{\boldsymbol{n}}. \tag{4.1}$$

With the complex-valued channel matrix $\overline{\boldsymbol{H}} \in \mathbb{C}^{\overline{M_R} \times \overline{M_T}}$, the receive vector $\overline{\boldsymbol{y}} \in \mathbb{C}^{\overline{M_R}}$, and the transmit vector $\overline{\boldsymbol{x}} \in \mathbb{C}^{\overline{M_T}}$. The noise vector $\overline{\boldsymbol{n}}$ consists of $\overline{M_R}$ i.i.d. complex Gaussian entries such that $\overline{\boldsymbol{n}} = \mathcal{CN}_{\bar{M}_R}(0, \sigma^2)$. To simplify the hardware design, and since most machine learning frameworks are not capable of processing complex numbers, the channel matrix can be decomposed into a real-valued channel matrix as follows:

$$\boldsymbol{H} = \begin{bmatrix} \Re\left(\overline{\boldsymbol{H}}\right) & -\Im\left(\overline{\boldsymbol{H}}\right) \\ \Im\left(\overline{\boldsymbol{H}}\right) & \Re\left(\overline{\boldsymbol{H}}\right) \end{bmatrix} \tag{4.2}$$

The functions $\Re(\cdot)$ and $\Im(\cdot)$ respectively obtain a complex number's real and imaginary parts. In a similar fashion, $\overline{\boldsymbol{y}}$ is decomposed to $\boldsymbol{y} \in \mathbb{R}^{M_R}; M_R = 2 \times \overline{M_R}$, $\overline{\boldsymbol{x}}$ is decomposed to $\boldsymbol{x} \in \mathbb{R}^{M_T}; M_T = 2 \times \overline{M_T}$, and $\overline{\boldsymbol{n}}$ is decomposed to $\boldsymbol{n} \in \mathbb{R}^{M_R}; M_R = 2 \times \overline{M_R}$. Please note that for simplicity the values $\boldsymbol{y}, \boldsymbol{H}, \boldsymbol{x}$, and $\boldsymbol{n}$ are redefined as real-value decomposed as compared to Eq. (2.12) where these quantities are complex-valued. The real-valued system model thus is described as

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{n}. \tag{4.3}$$

Due to its relative simplicity, the PGD algorithm is a natural choice for iteratively finding the solution for the massive MIMO ML detection problem of Eq. (2.15):

$$\widehat{\boldsymbol{x}}_{ML} = \underset{\boldsymbol{x} \in \mathcal{C}_C^{M_T}}{\arg\min} \ \|\boldsymbol{y} - \boldsymbol{H}\boldsymbol{x}\|_2^2 \tag{4.4}$$

The PGD algorithm is iterative in nature and has the form of [25]:

$$\widehat{\boldsymbol{x}}_{k+1} = \Pi\left[\widehat{\boldsymbol{x}}_k \; - \; \delta_k \frac{\partial \|\boldsymbol{y}-\boldsymbol{H}\boldsymbol{x}\|_2^2}{\partial \boldsymbol{x}}\bigg|_{\boldsymbol{x}=\widehat{\boldsymbol{x}}}\right], \tag{4.5}$$

$$= \Pi\left[\widehat{\boldsymbol{x}}_k \; - \; \delta_k(\boldsymbol{H}^T\boldsymbol{H}\widehat{\boldsymbol{x}}_k - \boldsymbol{H}^T\boldsymbol{y})\right], \tag{4.6}$$

$$= \Pi\left[\widehat{\boldsymbol{x}}_k \; - \; \delta_k\boldsymbol{H}^T(\boldsymbol{H}\widehat{\boldsymbol{x}}_k - \boldsymbol{y})\right], \tag{4.7}$$

where $\Pi[\cdot]$ is a projection operation onto the set of possible solutions in ${\mathcal{C}_{\mathcal{C}}}^{M_T}$. Note that the optimisation problem formalised here is different from the MMSE optimisation problem in Eq. (B.9). Here, directly the term $\|\boldsymbol{y}-\boldsymbol{H}\boldsymbol{x}\|_2^2$ is minimized by iteratively finding $\boldsymbol{x}$, whereas, in the MMSE case, an optimal matrix $\boldsymbol{W}$ is to be found which minimizes $\|\boldsymbol{x}-\boldsymbol{y}\boldsymbol{W}\|_2^2$. The projection operation represents another optimisation problem

$$\Pi(\boldsymbol{x}) = \underset{\boldsymbol{z}\in{\mathcal{C}_{\mathcal{C}}}^{M_T}}{\arg\min}\frac{1}{2}\|\boldsymbol{z}-\boldsymbol{x}\|_2^2. \tag{4.8}$$

It finds the closest value belonging to the symbol alphabet in terms of the norm2 distance. This operation, however, can be computationally highly complex due to the potentially high dimensionality of $\boldsymbol{x}$ and a potentially large constellation set $\mathcal{C}_{\mathcal{R}}$. A standard method for solving this problem exactly with reasonable computational complicity is sphere decoding [103]. However, a non-linear, element-wise projection operation is typically used, such as the signum function for BPSK and QPSK modulation.

Since the projection operation and the setting of the correct stepsize is non-trivial, one of the first proposals for ML-enhanced detectors for massive MIMO has been based on PGD and was named **DetNet** [25, 104]. In DetNet, the iterations of the PGD algorithm are unrolled into layers. As seen in Eq. (4.6), the terms $\boldsymbol{H}^T\boldsymbol{y}$ and $\boldsymbol{H}^T\boldsymbol{H}$ are essential for the PGD algorithm and therefore are provided as inputs to each layer. Further, $\widehat{\boldsymbol{x}}_k$ is provided as an input and

$\widehat{\boldsymbol{x}}_{k+1}$ as an output of each layer. Another parameter $\widehat{\boldsymbol{v}}_k$ is passed between layers. All inputs are concatenated, multiplied with a weight matrix and before going into a ReLU, the bias vector is added. To generate the outputs of the layer, $\widehat{\boldsymbol{x}}_k$ and $\widehat{\boldsymbol{v}}_k$ are multiplied by another weight matrix, and a bias vector is added for each output. DetNet showed good detection performance and inspired much research to follow up and improve upon the idea. A citation tree showing the most important connections of derived work is shown in Fig. 4.1. Apart from providing more results on DetNet, the work in [104] also introduces the FullyCon network. It is a simple neural network structure which does not directly take $\boldsymbol{H}$ into account. The results show that FullyCon works well for non-varying channels (i.e. training on each channel realisation is required).

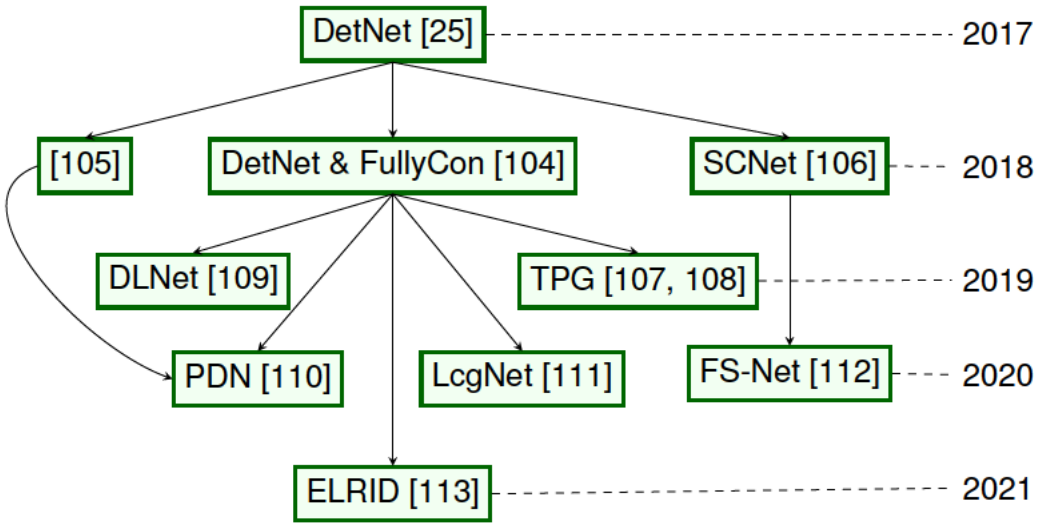


Figure 4.1: Overview of research based on PGD. The arrows indicate the most important citations and the year of publication on the right-hand side.

The work in [106] introduces a simplified version of DetNet called Sparsely Connected Net (**SCNet**). SCNet reduces the number of computations by a.) removing the secondary input/output signal $\widehat{\boldsymbol{v}}_k$, b.) removing unnecessary network connections (i.e. making the weight matrices sparse), and by c.) simplifying the loss function. It also claims better performance than DetNet on the tested fast-fading channel when a large number of antennas are present at the receiver and transmitter.

Another network simplification of SCNet is introduced in [112] and named: fast convergence, sparsely connected detection network (**FS-Net**). It proposes to share the same weight and bias values for the linear part of Eq. (4.6) ($\boldsymbol{H}^T\boldsymbol{H}\widehat{\boldsymbol{x}}_k - \boldsymbol{H}^T\boldsymbol{y}$) to simplify the network. Also, a different loss function is chosen to speed up training.

Inspired by DetNet, the work in [105] proposes a multilevel activation function which allows the network to be more efficient for higher modulation orders. Since DetNet chooses a softmax-type activation function to produce $\widehat{\boldsymbol{x}}$, it can only support BPSK and QPSK modulation types. In [105], the multilevel activation function is composed of multiple sigmoid functions and allows for higher modulation types in $\widehat{\boldsymbol{x}}_k$. Also, it is proposed to use multiple parallel networks, which are initialised via different initial guesses of $\widehat{\boldsymbol{x}}_0$ and select the final output depending on which network produced the smallest residual value in $\widehat{\boldsymbol{x}}_{M_I}$. This idea is further explored in the Parallel Detector Network (**PDN**) [110]. The authors demonstrate performance improvements over DetNet and test configurations of up to eight parallel detectors. This detector adds another term to the input vector of each layer, namely $\mathrm{diag}(\boldsymbol{H}^T\boldsymbol{H})\widehat{\boldsymbol{x}}_k$ resulting in the concatenated input of:

$$\boldsymbol{i}_{k+1} = \begin{bmatrix} \widehat{\boldsymbol{x}}_k - \delta_1\boldsymbol{H}^T\boldsymbol{H}\widehat{\boldsymbol{x}}_k - \delta_2\boldsymbol{H}^T\boldsymbol{y} \\ \mathrm{diag}(\boldsymbol{H}^T\boldsymbol{H})\widehat{\boldsymbol{x}}_k \\ \boldsymbol{v}_k \end{bmatrix}, \tag{4.9}$$

where $\boldsymbol{v}_k$ is $\boldsymbol{i}_{k+1}$ multiplied with a weight matrix. The concatenated, intermediate variable $\boldsymbol{i}_k$ is multiplied with another weight matrix and passed through an activation function and some other blocks to produce $\widehat{\boldsymbol{x}}_{k+1}$. The results show that even if only one detector is used, the detection performance over DetNet is improved.

Another approach with multiple, parallel FullyCon networks is presented in [113] and named **ELRID**. The combination of the outputs $\widehat{\boldsymbol{x}}_{M_I}$ of each FullyCon network is approached by summing the outputs as a weighted sum. It is not clear whether or not the weights for this summation are learned. However, in the next step, the resulting estimation of the transmitted symbol vector is processed in another PGD iteration. The work does not compare performance to the regular DetNet or FullyCon detector.

The detector network proposed in [109] is named Deep Learning Net (**DLNet**), and the layer structure is very similar to that of DetNet. However, the authors use 50 layers in their deep detector to improve detection performance. To manage the vanishing gradient problem in deep neural networks, each layer's input of $\widehat{\boldsymbol{x}}_k$ is weighted and added to the layer's output $\widehat{\boldsymbol{x}}_{k+1}$, such that $\widehat{\boldsymbol{x}}_{k+1} = \alpha\widehat{\boldsymbol{x}}_{k+1} + (1-\alpha)\widehat{\boldsymbol{x}}_k$. The authors set the numerical value of the hyperparameter α to 0.03 in their experiments and results, most likely from hyperparameter tuning in their environment.

The Trainable Projected Gradient (**TPG**) detector proposed in [107, 108] uses the compact form of the PGD algorithm as seen in Eq. (4.7). It replaces $\boldsymbol{H}^T$ in Eq. (4.7) by a matrix $\boldsymbol{W}$. This Matrix is set to the linear MMSE matrix: $\boldsymbol{W} \widehat{=} \boldsymbol{H}^T(\boldsymbol{H}\boldsymbol{H}^T + \alpha\boldsymbol{I}_{\boldsymbol{R}})^{-1}$. The parameter $\alpha \in \mathbb{R}$ is learnable, as well as other real-valued, scalar parameters in the linear part and the activation function of the detector. Since all learnable parameters are scalars, the number of learnable parameters is low. However, the detector performs well in comparison to other traditional detectors tested. A comparison between TPG, DetNet or any machine learning-enhanced algorithm is not provided.

The work in [111] introduces Learned Conjugate Gradient descent Net (**LcgNet**) uses a conjugate gradient descent algorithm instead of the PGD algorithm. This algorithm targets faster convergence by ensuring directions of the descent are orthogonal to each other. Results are provided for various antenna configurations and for a modulation scheme of up to QAM16. A detailed comparison with DetNet shows improvements in detection performance and lower computational complexity.

The PGD based detectors achieve good performance and are popular due to their conceptual simplicity. However, many PGD approaches use many layers and might be computationally complex. DetNet, for example, uses 3000 layers and DLNet uses 50 layers [25] [109]. AMP based detectors have been proposed as an alternative and are reviewed next.

4.2.3 Orthogonal Approximate Message Passing (OAMP)

Classical AMP and OAMP detection

AMP has been inspired by iterative thresholding algorithms and has been first conceived for compressed sensing applications [114, 115]. Since then, variations of the algorithm have been applied to various statistical estimation tasks such as machine learning, image processing, and communications [116]. The basic form of the AMP algorithm is:

$$\boldsymbol{r}_k = \boldsymbol{y} - \boldsymbol{H}\widehat{\boldsymbol{x}}_k + \boldsymbol{v}_k, \tag{4.10}$$

$$\boldsymbol{z}_k = \boldsymbol{H}^T\boldsymbol{r}_k + \widehat{\boldsymbol{x}}_k, \tag{4.11}$$

$$\widehat{\boldsymbol{x}}_{k+1} = \eta(\boldsymbol{z}_k), \tag{4.12}$$

where $\boldsymbol{r}_k$ is the residual, $\boldsymbol{z}_k$ is a noisy estimate of $\boldsymbol{x}_{k+1}$, and $\boldsymbol{v}_k$ is an Onsager term. It is worth noting that the linear part of PGD (i.e. the argument in Eq. (4.7)) is equivalent to $\boldsymbol{z}_k$ in Eq. (4.11), except the omission of the step size δ_k in Eq. (4.11) and the addition of the Onsager term. The Onsager term is what differentiates the AMP algorithm from the iterative thresholding algorithm.

Eq. (4.12) differs from PGD as the noisy estimate $\boldsymbol{z}_k$ is denoised element-wise by the denoiser function $\eta(\cdot)$ in Eq. (4.12).

Based on the AMP algorithm and its application in compressed sensing, the authors in [117] propose the Individually Optimal LAMA (IO-LAMA) algorithm. The same algorithm is also just called LArge MIMO AMP (LAMA) and is shown in Algorithm 2 [118, 119].

Algorithm 2: LAMA algorithm [117].

input : Received signal $\boldsymbol{y} \in \mathbb{R}^{M_R}$
input : Channel matrix $\boldsymbol{H} \in \mathbb{R}^{M_R \times M_T}$
input : Noise variance $\sigma^2 \in \mathbb{R}$
output: $\widehat{\boldsymbol{x}}_{I+1} \in \mathbb{R}^{M_T}$

/* initialization */
$\widehat{\boldsymbol{x}}_1 \in \mathbb{R}^{M_T} \leftarrow \mathbb{E}_X[\mathcal{C}_\mathcal{R}]$
/* preprocessing */
$\beta \leftarrow M_T/M_R;\ \boldsymbol{y}_{MRC} \leftarrow \boldsymbol{H}^T\boldsymbol{y};\ \boldsymbol{G}_T \leftarrow \boldsymbol{H}^T\boldsymbol{H}$
/* iteration loop */
for $i \leftarrow 1$ **to** M_I **do**
  $\boldsymbol{z}_i \leftarrow \boldsymbol{y}_{MRC} + (\boldsymbol{I_T} - \boldsymbol{G_T})\widehat{\boldsymbol{x}}_i + \boldsymbol{v}_i$ // linear function
  $\widehat{\boldsymbol{x}}_{i+1} \leftarrow \mathrm{F}(\boldsymbol{z}_i, \sigma^2/2 + \beta\tau_i)$ // calculate estimate of $\widehat{\boldsymbol{x}}$
  $\boldsymbol{\phi}_{i+1} \leftarrow \mathrm{G}(\boldsymbol{z}_i, \sigma^2/2 + \beta\tau_i)$ // calculate estimate of variance $\boldsymbol{\tau}$
  $\tau_{i+1} \leftarrow \sum_{k=1}^{M_T} \boldsymbol{\phi}_{i,k}$
  $\boldsymbol{v}_{i+1} \leftarrow \frac{\beta\tau_{i+1}}{\sigma^2/2+\beta\tau_i}(\boldsymbol{z}_i - \widehat{\boldsymbol{x}}_i)$ // calculate new onsager term
end

The function $\mathrm{F}(\boldsymbol{z}, \boldsymbol{\tau})$ produces the mean value of $\boldsymbol{z}_i$ (i.e. the denoised, new estimate of $\widehat{\boldsymbol{x}}_{i+1}$). The function $\mathrm{G}(\boldsymbol{z}, \boldsymbol{\tau})$ produces the variance of $\boldsymbol{z}$. If viewed from a message-passing perspective, these functions produce the mean and variance of the messages, hence the algorithm name. The functions are defined as

$$\mathrm{F}(\boldsymbol{z}_k, \boldsymbol{\tau}) = \int_{c \in \mathcal{C}_\mathcal{C}} c \times f(c|\boldsymbol{z}_k, \tau) dc, \tag{4.13}$$

$$\mathrm{G}(\boldsymbol{z}_k, \boldsymbol{\tau}) = \int_{c \in \mathcal{C}_\mathcal{C}} |c|^2 \times f(c|\boldsymbol{z}_k, \tau) dc - |\mathrm{F}(\boldsymbol{z}_k, \boldsymbol{\tau})|^2, \tag{4.14}$$

where $f(c|\boldsymbol{z}_k, \tau)$ is the posterior PDF defined as

$$f(c|\boldsymbol{z}_k, \tau) = \frac{1}{Z} p(\boldsymbol{z}_k|c, \tau) p(c); \text{with } p(\boldsymbol{z}_k|c, \tau) \approx \mathcal{RN}(c, \tau), \tag{4.15}$$

with Z being a normalisation constant. Since we assume equiprobable symbols in $\boldsymbol{x}$, $p(c)$ can be omitted. The core observation is that in AMP, the noise on $\boldsymbol{z}$ is i.i.d Gaussian distributed under the condition that the channel matrix $\boldsymbol{H}$ and the noise $\boldsymbol{n}$ is i.i.d Gaussian distributed [118]. If these assumptions hold, then the ideal implementation of the mean- and variance- functions F and G are given as shown in Algorithm 3.

Algorithm 3: Mean and Variance functions F and G

input : noisy estimate $\boldsymbol{z} \in \mathbb{R}^{M_T}$
input : noise variance $\boldsymbol{\tau} \in \mathbb{R}$
input : unique symbol values $\boldsymbol{c} \in \mathcal{C}_{\mathcal{R}}$
output: denoised estimate $\widehat{\boldsymbol{x}} \in \mathbb{R}^{M_T}$
output: noise estimate $\boldsymbol{\phi} \in \mathbb{R}^{M_T}$

```
for j ← 1 to M_T do                                   // row-wise
    for k ← 1 to M_C do                               // column-wise
        arg_{j,k} ← −(z_j − c_k)^2 / (2×τ)            // arg ∈ R^{M_T × M_C}
    end
end
foreach a ∈ arg do  a ← e^a                           // element-wise
for j ← 1 to M_T do                                   // row-wise
    ssum ← (Σ_{k=1}^{M_C} arg_{j,k})^{-1}             // ssum ∈ R
    for k ← 1 to M_C do                               // column-wise
        prob_{j,k} ← arg_{j,k} × ssum
    end
end
x ← prob × c;  φ ← prob × c^2 − x^2
```

Another often-used variant of the original AMP algorithm to perform massive MIMO detection is OAMP [120]. It was introduced to improve the stability of the classical AMP algorithm for channel matrices not following an i.i.d Gaussian distribution. The OAMP algorithm is shown in Algorithm 4. It can be implemented in different flavours, allowing performance and computational complexity to be traded off. By setting $\boldsymbol{W}_i$ to one of the classical linear detector matrices $\boldsymbol{W}_{MRC}$, $\boldsymbol{W}_{ZF}$, or $\boldsymbol{W}_{MMSE}$ as introduced in Section 2.3.2 this can be realised.

In the case of setting $\boldsymbol{W}_i$ to $\boldsymbol{W}_{MRC}$ or $\boldsymbol{W}_{ZF}$ the computational complexity can be reduced since $\boldsymbol{W}_i$, $\boldsymbol{B}$, tr_B, and tr_W can be calculated during preprocessing and are the same for all iterations. This setting, however, will reduce performance. The work in [120] finds an optimal matrix to be used in the algorithm, namely $\boldsymbol{W}_{opt}$. This is the most complex option and needs to be computed in each iteration on top of the MMSE computation. In particular, the matrix inverse might be prohibitively complex for real-time deployment.

To understand the OAMP algorithm, a concise derivation of τ_i and v_i is provided. For a rigorous derivation, the interested reader is referred to [120]. First, two error terms are defined as

$$\boldsymbol{h}_i = \boldsymbol{z}_i - \boldsymbol{x} \quad \texttt{//Error in the linear estimation}, \tag{4.16}$$

$$\boldsymbol{q}_i = \widehat{\boldsymbol{x}}_i - \boldsymbol{x} \quad \texttt{//Error in denoised estimation}. \tag{4.17}$$

By using Line 9 of Algorithm 4 and Eq. (4.17), $\boldsymbol{h}_i$ can be expressed as

$$\boldsymbol{h}_i = \boldsymbol{q}_i + \boldsymbol{x} + \boldsymbol{W}_i\boldsymbol{y} - \boldsymbol{W}_i\boldsymbol{H}(\boldsymbol{q}_i + \boldsymbol{x}) \tag{4.18}$$

$$= \boldsymbol{q}_i \underbrace{(\boldsymbol{I_T} - \boldsymbol{W}_i\boldsymbol{H})}_{\boldsymbol{B}} + \boldsymbol{W}_i \underbrace{(\boldsymbol{y} - \boldsymbol{H}\boldsymbol{x})}_{\boldsymbol{n}} \tag{4.19}$$

The MSE of the error of the linear estimation $\boldsymbol{h}_i$ (Eq. (4.16)) can be expressed as [121]

$$v_i = \mathrm{MSE}(\boldsymbol{q}_i) = \frac{1}{M_T}\,\mathbb{E}\left\{||\boldsymbol{q_i}||_2^2\right\} = \frac{\|r_i\|_2^2 - M_R\sigma^2}{tr_G}. \tag{4.20}$$

The MSE of the error in the noisy estimation $\boldsymbol{h}_i$ (Eq. (4.17)) can be expressed by using Eq. (4.19) and Eq. (4.20) as

Algorithm 4: OAMP algorithm with selectable OAMP matrix type W_{type} [120].

input : Received signal $\boldsymbol{y} \in \mathbb{R}^{M_R}$
input : Channel matrix $\boldsymbol{H} \in \mathbb{R}^{M_R \times M_T}$
input : OAMP matrix type W_{type}
input : Noise variance $\sigma^2 \in \mathbb{R}$
output: $\widehat{\boldsymbol{x}}_{M_I+1} \in \mathbb{R}^{M_T}$

/* initialization */
$\widehat{\boldsymbol{x}}_1 \in \mathbb{R}^{M_T} \leftarrow \mathbb{E}[\mathcal{C}_{\mathcal{R}}]$; $\tau_0 \leftarrow 1$
/* preprocessing */
if $W_{type} = mrc$ **then** $\boldsymbol{W}_i \leftarrow \boldsymbol{H}^T; \forall i \in [0, M_I]$
else if $W_{type} = pinv$ **then** $\boldsymbol{W_i} \leftarrow (\boldsymbol{H}^T\boldsymbol{H})^{-1}\boldsymbol{H}^T; \forall i \in [0, M_I]$
$\boldsymbol{B} \leftarrow \boldsymbol{I_T} - \boldsymbol{W}_i\boldsymbol{H}$; $\boldsymbol{G_T} \leftarrow \boldsymbol{H}^T\boldsymbol{H}$
$tr_G \leftarrow \mathrm{Tr}(\boldsymbol{G_T})$; $tr_B \leftarrow \mathrm{Tr}(\boldsymbol{B}^T\boldsymbol{B})$
$tr_{Wi} \leftarrow \mathrm{Tr}(\boldsymbol{W}_i^T\boldsymbol{W}_i); \forall i \in [0, M_I]$
/* detector loop */
for $i \leftarrow 1$ **to** M_I **do**
  /* linear function */
  $\boldsymbol{r}_i \leftarrow \boldsymbol{y} - \boldsymbol{H}\widehat{\boldsymbol{x}}_i$
  $\boldsymbol{z}_i \leftarrow \widehat{\boldsymbol{x}}_i + \boldsymbol{W}_i\boldsymbol{r}_i$
  /* calculate new estimate of $\widehat{\boldsymbol{x}}$ */
  $\widehat{\boldsymbol{x}}_{i+1} \leftarrow \mathrm{F}(\boldsymbol{z}_i, \boldsymbol{\tau}_i)$
  /* calculate new estimate of variance $\boldsymbol{\tau}$ */
  $v_i \leftarrow \frac{\|r_i\|_2^2 - M_R\sigma^2}{tr_G}$
  if $W_{type} = opt$ *or* $W_{type} = mmse$ **then**
    $\hat{\boldsymbol{W}}_{\boldsymbol{i}} \leftarrow v_i\boldsymbol{H}^T(v_i\boldsymbol{H}\boldsymbol{H}^T + \frac{\sigma^2}{2}\boldsymbol{I_R})^{-1}$
  end
  if $W_{type} = mmse$ **then** $\boldsymbol{W}_i \leftarrow \hat{\boldsymbol{W}}_i$
  if $W_{type} = opt$ **then** $\boldsymbol{W}_i \leftarrow \frac{2M_T}{\mathrm{Tr}(\hat{\boldsymbol{W}}_i\boldsymbol{H})}\hat{\boldsymbol{W}}_i$
  if $W_{type} = opt$ *or* $W_{type} = mmse$ **then**
    /* overwrite B, tr_B and tr_W */
    $\boldsymbol{B} \leftarrow \boldsymbol{I_T} - \boldsymbol{W}_i\boldsymbol{H}$
    $tr_B \leftarrow \mathrm{Tr}(\boldsymbol{B}^T\boldsymbol{B})$; $tr_W \leftarrow \mathrm{Tr}(\boldsymbol{W}^T\boldsymbol{W})$;
  end
  $\tau_i \leftarrow \frac{1}{2M_T}tr_B \times v_i + \frac{1}{4M_T}tr_W \times \sigma^2$
end

$$\tau_i = \mathrm{MSE}(\boldsymbol{h}_i) = \frac{1}{M_T}\,\mathbb{E}\left\{||\boldsymbol{h_i}||_2^2\right\}, \tag{4.21}$$

$$= \frac{1}{M_T}\,\mathbb{E}\left\{\boldsymbol{q}_i\boldsymbol{B} + \boldsymbol{W}_i\boldsymbol{n}\right\}, \tag{4.22}$$

$$= \frac{\boldsymbol{v}_i}{M_T}\,\mathbb{E}\left\{\boldsymbol{B}\right\} + \frac{\sigma^2}{M_T}\,\mathbb{E}\left\{\boldsymbol{W}_i\right\}, \tag{4.23}$$

$$= \frac{\boldsymbol{v}_i}{M_T}\mathrm{tr}(\boldsymbol{B}) + \frac{\sigma^2}{M_T}\mathrm{tr}(\boldsymbol{W}_i), \tag{4.24}$$

and corresbonds to Line 21 in Algorithm 4.

ML-enhanced OAMP detection

For both AMP and OAMP algorithms, the performance of the algorithm is primarily determined by the noise distribution on $\boldsymbol{z}_i$. The linear part of the algorithm can be seen as shaping the noise as such by calculating the appropriate $\boldsymbol{W}_i$. Also, the accuracy of estimating the noise variance τ is decisive in determining the performance. The step from AMP to OAMP improves these two parameters for the practical application to MIMO and delivers higher detection performance. Another algorithm evolution is applying ML to learn and optimise these parameters to achieve even higher detection performance.

The authors in [31] propose to add learnable, scalar parameters to the OAMP algorithm to allow for finding the optimal step size. This detector is named **OAMP-Net**. In the linear function at Line 9 of Algorithm 4, the parameter $\boldsymbol{\theta_1}$ is added such that

$$\boldsymbol{z}_i \leftarrow \widehat{\boldsymbol{x}}_i + \theta_{1,i}\boldsymbol{W}_i\boldsymbol{r}_i. \tag{4.25}$$

Further, a parameter scaling the noise variance in τ is added in Line 21:

$$\tau_i \leftarrow \frac{1}{2M_T} tr_B \times v_i + \frac{\theta_{2,i}}{4M_T} tr_W \times \sigma^2. \tag{4.26}$$

In OAMP-Net, the matrix W_i is chosen to be the optimal matrix ($W_{type} = \mathrm{opt}$). In a previous implementation for sparse signal recovery applications, the pseudo-inverse of $\boldsymbol{H}$ is assigned to all W_i ($W_{type} = \mathrm{pinv}$). This network was called TISTA [122]. Another related work to OAMP-Net is the Trainable AMP (**TAMP**) detector presented in [123]. It consists of a preprocessing stage which calculates the Singular Value Decomposition (SVD) of the channel matrix. Next, an unrolled Generalized Approximate Message Passing (GAMP) detector is enhanced with learnable parameters to perform the signal detection. Simulation results show some performance improvement over OAMP-Net. However, the SVD decomposition in the preprocessing is also computationally expensive. A citation tree shows the most important connections of derived work in Fig. 4.2.

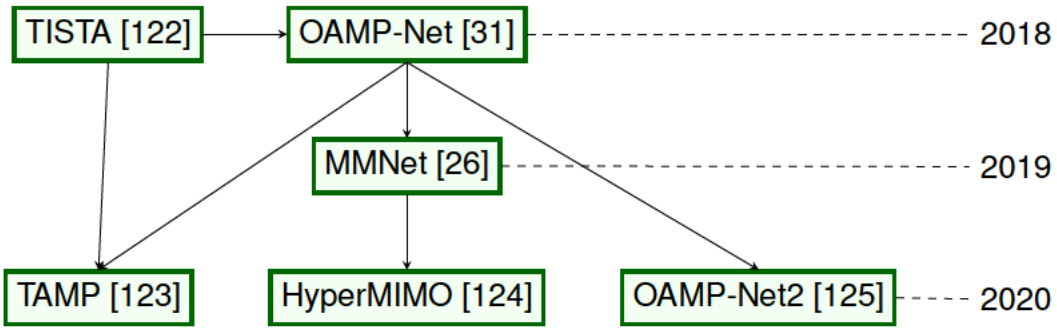


Figure 4.2: Overview of research based on approximate message passing, including the most important citations indicated by arrows and the year of publication on the right-hand side.

The work in [26] introduces other learnable parameters into the OAMP algorithm and presents two versions, namely **MMNet-iid** and **MMNet**. The former sets $\boldsymbol{W}_i = \boldsymbol{H}^T$ and introduces a learnable parameter in the linear part and noise variance estimation part similar to that of OAMP-Net. MMNet, on the other hand, introduces $\boldsymbol{W}_i$ as a learnable matrix and adds a vector to scale the noise estimate individually for each element of $\widehat{\boldsymbol{x}}$.

MMNet's performance on realistic channels is impressive. However, the cost for this performance is its requirement of online training. Online training means that before the detector can be used, it has to be trained on the specific realisation of the channel and retrained in every coherence interval. The training could be based on random, on-the-fly generated, $\boldsymbol{x}$-values and the current

channel matrix. If the detector is trained on a slightly different channel matrix, it performs very poorly, as shown later in Chapter 5. In practice, this means an excessively high detection latency as the detector's training must be completed first.

On the other hand, OAMP-Net performs well without online training. However, its computational complexity is challenging for real-time deployment. It requires the MMSE-matrix calculation with current noise estimates for each channel use and algorithm iteration. This includes a full matrix inverse in each iteration (Algorithm 4, Line 13).

To avoid the issue of online learning, the work in [124] introduces **Hyper-MIMO**. A hyper-network describes a setup where one neural network learns the parameters used in another neural network. Hence HyperMIMO extends MMNet by a fully connected neural network which learns the parameters of MMNet (such as $\boldsymbol{W}_i$) for all channel realisations. The hyper-network takes in the upper triangular matrix of the QR-decomposed channel matrix and consists of multiple dense neural network layers. The disadvantage of HyperMIMO is a slight loss in performance compared to MMNet and the higher computational cost. The QR decomposition of the channel matrix is costly; however, it must be performed only once per channel coherence interval. Further, the dense layers might require a large number of parameters for a large number of connected users. Unfortunately, HyperMIMO is evaluated only on a specific local scattering channel model. It is unclear whether or not the reported performance can be achieved using a realistic channel model such as QUAsi Deterministic RadIo channel GenerAtor (QuaDRiGa) [126].

The work in [125] introduces **OAMP-Net2**, which presents several improvements over OAMP-Net. Firstly, most previous detectors assume perfect CSI; however, in OAMP-Net2, possible CSI estimation errors are taken into account, thereby improving overall performance. Secondly, soft-output information is

provided, which is preferred by blocks coming after the detector. Thirdly, the set of learnable parameters has been updated. Overall, OAMP-Net2 shows improved performance and robustness as compared to OAMP-Net. However, a comparison with [26] is missing.

4.2.4 Other Notable Detectors

Belief Propagation

Another algorithm relying on message passing is Belief Propagation (BP). Conceptually, this algorithm consists of observational nodes and symbol nodes connected in a factor graph or a Markov random field [127]. Messages are iteratively passed between the nodes providing an efficient way of estimating $\widehat{\boldsymbol{x}}$. Classical BP detectors are proposed in [127, 128]. Dampening factors and other parameters can be learned and improve the detection performance. The detector proposed by the authors in [129] adds learnable dampening factors to the messages passed between nodes, thereby outperforming the classical BP algorithm. The tested MIMO configurations are 8x8 and 16x16 with a Rayleigh channel model and BPSK modulation. In [21], a damped BP, a max-sum BP, and a simplified Message Passing Detector (sMPD) algorithm are proposed, and each algorithm is extended with learnable parameters. The damped BP algorithm learns the damping factors. The max-sum algorithm learns damping, normalisation, and offset factors, and the sMPD algorithm learns damping, re-scaling, and offset factors. In the performance evaluation with antenna configurations of 32x8, 64x8, and 128x8, a Rayleigh channel, and a modulation type of QAM16, the machine-learning-enhanced detectors show better or equal performance as MMSE detection.

ADMM

The ADMM algorithm is commonly used to solve constraint convex optimisation problems and has been proposed for MIMO detection (e.g. [130]). Inspired by the deep unfolding and by the previous work on ML-enhanced PGD detectors, the work in [131] proposes to apply a similar strategy to the ADMM algorithm. The penalty parameter $\boldsymbol{\lambda} \in \mathbb{R}^{M_N}$ is split up into multiple parts, which are multiplied element-wise. One part is set to the per-user channel power, and the other is a learnable parameter. The authors name the resulting detector **ADMM-Net** and provide SER performance analysis for various antenna configurations. The authors claim a less complex detector structure and improved SER detection performance compared to the PGD-based DetNet detector. The proposed detector, however, only supports BPSK and QPSK modulation schemes.

The work in [132] removes this limitation by basing their detector on the recently proposed Penalty Sharing (PS)-ADMM algorithm [133]. The authors then propose to learn a penalty parameter similar to the work in [131] and name this detector **ADMM-PSNet**. In a further investigation, a MultiLayer Perceptron (MLP) is introduced to replace the expensive $\widehat{\boldsymbol{x}}$-estimation step, which involves a MMSE-matrix inversion in the PS-ADMM algorithm. This detector is named **ADMM-HNet** and reduces complexity significantly. Compared to DetNet and the traditional MMSE detector, improved detection performance is shown over a flat fading i.i.d Rayleigh channel.

As the research in the field of ML-based detection is vast and fast-evolving, only some recently proposed detectors can be reviewed here. Table 4.1 gives an overview of the computational complexity of various of the above discusses detectors. The first four rows show a large number of learnable parameters in model-driven designs. DetNet can reduce this number; however, is still computationally complex. Promising detectors from a complexity perspective are based on partial learning, OAMP, or further simplified PGD detectors. A recent survey specifically on uplink massive MIMO detectors can be found [134] and might interest the reader.

Table 4.1: Complexity comparison of various MIMO or massive MIMO detectors in terms of multiplications used per detection.

Detector	Comment	Number of Multiplications
DenseNet [99]	Repurposed image classification networks	$312k \times M_R \times M_T$ (1)
MobileNetV2 [99]		$255k \times M_R \times M_T$ (1)
CNN [101]	2x2 MIMO	215k(1)
DNN [101]	2x2 MIMO	208k(1)
Partial Learning [102]	64x64 MIMO	$\approx 49k$ + traditional detector overhead
DetNet [25, 104]	Based on PGD algorithm	$3M_T$ Layers $\times(52M_T^2 + 11M_T)$
OAMPNet [31] $\boldsymbol{W}_i = \boldsymbol{H}^T$	A detailed complexity analysis can be found in Chapter 5	$M_I(5M_TM_C + 2M_RM_T + M_R + 5)$
MMNet [26] $\boldsymbol{W}_i$ is learned		$M_I(5M_TM_C + 2M_RM_T + M_R + 5)$
(1) Number of learnable parameters, not multiplications		

4.2.5 The importance of channel models

In the diverse proposals of ML-enhanced detectors, various MIMO channel models are used. These models range from simply drawing i.i.d. samples from a Gaussian distribution for each entry of $\boldsymbol{H}$ to sophisticated, geometry-based stochastic channel models such as QuaDRiGa [126]. The channel model is of undoubted importance for the performance estimation of traditional detectors. Even more so for ML-based detectors as the models learn the channel characteristics by optimising their parameters accordingly. In Table 4.2, the previously introduced detectors are classified by the channel model used for performance evaluation. It can be seen that most designs rely on simple sampling from an i.i.d Gaussian or i.i.d Rayleigh distribution. This is sub-optimal as it does not allow performance estimation of an actual deployment scenario. The difference between an i.i.d Gaussian channel and a real deployment scenario can be very significant, as the example of the classical AMP and LAMA algorithm shows. It performs well on the i.i.d Gaussian channel since the noise at the denoiser input will be i.i.d Gaussian shaped as well; however, it performs very poorly

on realistic channels (see Section 5.4). The Toeplitz- and Kronecker-matrix-based channels introduce some correlation between base station antennas, providing a slightly advanced model. The QuaDRiGa model is a statistical ray-tracing model capable of simulating three-dimensional propagation based on statistical geometrical modelling of scattering clusters. One of the few proposed ML-based detectors which are simulated with a realistic channel model as QuaDRiGa is MMNet [26]. This is one of the reasons why OAMP-based detectors are investigated closer below.

Table 4.2: Channel models used for SER performance simulations in the literature.

Channel Model					
i.i.d. Gaussian	i.i.d. Rayleigh	Toeplitz	Kronecker [135]	QuaDRiGa [126]	Others
[26, 25, 104, 109, 110, 112, 113, 131]	[21, 31, 100, 111, 123, 125, 129, 132]	[104, 105]	[123, 125]	[26, 136]	[99] Nakagami-m [25] Ill cond. mat. [101] Rician channel [124] Local scattering

In a typical deployment scenario, the channel matrix has to be obtained by channel estimation techniques. These estimation techniques cannot recover the exact channel matrix; however, most work in the literature assumes perfect CSI for training their detectors. Ideally, CSI estimation errors are also considered (e.g. [101],[125]). Joint channel estimation and signal detection approach could even perform better, which is one of the benefits of applying the ML techniques (e.g. [99]).

4.2.6 Real-time Processing

As seen above, many different ML enhanced detector algorithms are proposed in the literature. Typically they evaluate the proposed algorithms only in terms of SER detection performance in abstract, link-level simulations. Some research reports computational complexity, but most do not consider real-time computation. This can be explained by the fact that the field of machine learning in the physical layer of communication systems is fairly new and just emerging. Some work has been undertaken to show real-time processing capability and will be reviewed next.

The authors in [21] provide an ASIC implementation of their sMPD algorithm (see Section 4.2.4). The supported MIMO configuration is 128x8 with a QAM16 modulation. The accelerator is implemented based on a $65\,\mathrm{nm}$ Complementary Metal-Oxide-Semiconductor (CMOS) process and achieves a throughput of $\approx 180\,\mathrm{Mbit/s}$. Compared to previous, non-machine learning enhanced implementations, the proposed design achieves significantly lower normalised throughput and area efficiency. The authors attribute this not to the machine learning enhancements but to the optimised design methodology and the hard decision output of the compared work. The work is an important step towards real-time deployment; however, a shortcoming is the used channel model of Rayleigh fading. It is not a realistic channel model, and it is unclear how the proposed detector performs on other channels. Further, many hardware details are missing, and implementation details regarding the machine learning parts are unclear. Finally, the hardware only supports one configuration in terms of antenna/user setup and modulation type.

The work in [23] develops a machine learning MIMO-OFDM detector based on an ESN. This type of neural network generates reservoir states and connections between them. The reservoir states and the mapping from the input to the states are generated randomly; mealy, the weights mapping from the reservoir to the output are trained. According to the authors, the fast training on limited training data of the ESN makes this approach especially suitable for a communications problem. The specific scenario is not further described. The work targets Wi-Fi frames with fixed pilot positions in frequency (i.e. comb-type) [137]. A MIMO configuration of 4x4 with 64 subcarriers is used. The modulation type is QAM16, and the channel is created following the 3GPP extended pedestrian model-A with a user speed of 10 km/h. For testing, the authors utilise a SDR platform with an FPGA for the acceleration of the ESN workload. During over-the-air testing, the proposed detector outperformed LMMSE symbol detection in non-LOS scenarios in terms of BER. The transmission data rate, and the accelerator design including throughput and latency numbers are not specified clearly. Also, the scalability of the approach to massive MIMO applications is not addressed in the work.

Another real-time MIMO detector proposal is made in [136]. The authors propose a joint channel decoding and signal detection scheme based on **TurboNet** [138] and their Expectation Propagation (EP) based decoder called **EPNet**. TurboNet utilizes a turbo decoder for channel decoding and exchanges Log-Likelihood Ratio (LLR)s with the EPNet decoder. Based on this, the EPNet decoder calculates a new *a priori* LLR which is provided as input to the next TurboNet stage. Multiple stages are linked together resulting in an unrolling of the iterative detection process. In simulations, a MIMO system of up to 64×32 with a QAM4 modulation is evaluated. The channel matrix used for comparison with other detectors either follows a Rayleigh distribution or the 3GPP 3D correlated channel as in [26]. The results show that EPNet (without TurboNet) outperforms MMNet and OAMP-Net in the correlated channel case. OAMP-Net and MMNet show approximately the same SER, which is surprising given that typically MMNet outperforms OAMP-Net considerably (see Section 5.4). The authors also provide over-the-air experimental results in a 12x8 antenna setup where the receiver is implemented in non-real-time software. Even though the processing is not performed in real-time, the over-the-air transmission is an important step towards that goal.

As described above, some attempts at processing ML enhanced detection algorithms in real-time have been made recently. However, these attempts do not focus on computation and hardware. It is often not clear what the real-time properties - such as maximum throughput, latency, and required computing resources - of these systems are. These are essential parameters to enable the deployment of the proposed algorithms in actual real-time systems. Further, the profiling across system parameters is not provided. Real-time behaviour across various MIMO configurations such as the number of BS antennas, number of UEs, modulation type, etc. is also not discussed.

4.3 An ML-Enhanced OAMP-type Accelerator

This section proposes a general ML-enhanced OAMP-type accelerator. It supports multiple detector types such as MMNet, OAMP-Net and the classical OAMP algorithm. The implemented detector type is configurable by providing the according parameters from outside the detector. This proposal will first assess the feasibility of ML-enhanced OAMP detection from a processing perspective. As introduced above, OAMP-type algorithms have been intensely researched for being expanded with ML enhancements. However, the feasibility of processing them in real time has not been assessed. The OAMP algorithm and, even more so, the AMP algorithm are sensitive to the statistical characteristics of the channel matrix $\boldsymbol{H}$. The learned parameters in ML-enhanced detectors can, in fact, stabilise the algorithm and remove this constraint. For this reason, OAMP-type detectors benefit significantly from the ML technique and, thereby, are worth investigating for real-time deployment. Further, regarding the decision on which detector shall be investigated for real-time deployment, OAMP-type detectors were a good choice as MMNet was one of the few detectors evaluated on realistic channel models at that time. Realistic channel models are essential, as elaborated in Section 4.2.5.

In this chapter, we propose to extend Algorithm 4 with machine learning parameters to form Algorithm 5. This algorithm can be seen as a combination of MMNet and OAMPNet and can implement both. The learnable parameters are highlighted by light-green background colour. Different detectors can be implemented by assigning different values to each learnable parameter. The three-dimensional matrix $\boldsymbol{\Theta}$ is learnable, provides an $M_T \times M_R$ matrix for each iteration of the algorithm, and is only used in the MMNet case. The learnable parameter $\boldsymbol{\Lambda}$ provides one scalar per iteration and can be interpreted as a learnable step size by which the contribution of the current iterations estimation of $\widehat{\boldsymbol{x}}$ to the noisy symbol estimate $\boldsymbol{z}_i$ is regulated. It is only useful in cases where $\boldsymbol{W}_i$ is not a learnable matrix (i.e. not in MMNet). The $\boldsymbol{\nu}$ parameters are used to scale the noise variance estimate τ_i. Some algorithms introduce separate scaling for the individual terms when calculating τ; however, sometimes $\boldsymbol{\nu}^{\mathbf{1}}$ and

$\boldsymbol{\nu^2}$ will be set to the same value. The configuration of the learned parameters for each detector type is shown in Table 4.3. When comparing Algorithm 5 with Algorithm 4, it is seen that only configurations of W_{type} are supported, which do not require recalculating the matrices from preprocessing (i.e. $W_{type} = \text{mmse}$ and $W_{type} = \text{opt}$ are not supported). This is done to keep the accelerator design reasonably simple for a first comparison. A proposal for implementing the types for which some preprocessing matrices are recalculated is given in Chapter 5.

Algorithm 5: Machine learning extended OAMP accelerator algorithm. Variables with light-green background colour highlight the learnable parameters $\boldsymbol{\Theta}$, $\boldsymbol{\Lambda}$, $\boldsymbol{\nu}^1$, and $\boldsymbol{\nu}^2$.

input : Received signal $\boldsymbol{y} \in \mathbb{R}^{M_R}$
input : Channel matrix $\boldsymbol{H} \in \mathbb{R}^{M_R \times M_T}$
input : OAMP matrix type W_{type}
input : Noise variance $\sigma^2 \in \mathbb{R}$
input : Learned parameters $\boldsymbol{\Theta} \in \mathbb{R}^{M_I \times M_T \times M_R}; \boldsymbol{\Lambda} \in \mathbb{R}^{M_I}; \boldsymbol{\nu}^1, \boldsymbol{\nu}^2 \in \mathbb{R}^{M_I}$
output: $\widehat{\boldsymbol{x}}_{M_I} \in \mathbb{R}^{M_T}$

/* initialization */
$\widehat{\boldsymbol{x}}_1 \in \mathbb{R}^{M_T} \leftarrow \mathbb{E}[\mathcal{C}_{\mathcal{R}}]$; $\tau_0 \leftarrow 1$
/* preprocessing */
if $W_{type} = \boldsymbol{H^T}$ **then** $\boldsymbol{W}_i \leftarrow \boldsymbol{H^T}; \forall i \in [0, M_I]$
else if $W_{type} = pinv(\boldsymbol{H})$ **then** $\boldsymbol{W}_i \leftarrow \text{pinv}(\boldsymbol{H}); \forall i \in [0, M_I]$
else if $W_{type} = MMNet$ **then** $\boldsymbol{W} \leftarrow \boldsymbol{\Theta}$
$\boldsymbol{B}_i \leftarrow \boldsymbol{I_T} - \boldsymbol{\Lambda}_i \boldsymbol{W}_i \boldsymbol{H}; \forall i \in [0, M_I]$; $\boldsymbol{G}_T \leftarrow \boldsymbol{H}^T \boldsymbol{H}$
$tr_G \leftarrow \text{Tr}(\boldsymbol{G}_T)$; $tr_{Bi} \leftarrow \text{Tr}(\boldsymbol{B}_i^T \boldsymbol{B}_i)$
$tr_{Wi} \leftarrow \text{Tr}(\boldsymbol{W}_i^T \boldsymbol{W}_i); \forall i \in [0, M_I]$
/* detector loop */
for $i \leftarrow 1$ **to** M_I **do**
  /* linear function */
  $\boldsymbol{r}_i \leftarrow \boldsymbol{y} - \boldsymbol{H}\widehat{\boldsymbol{x}}_i$
  $\boldsymbol{z}_i \leftarrow \widehat{\boldsymbol{x}}_i + \boldsymbol{\Lambda}_i \times \boldsymbol{W}_i \boldsymbol{r}_i$
  /* calculate new estimate of $\widehat{\boldsymbol{x}}$ */
  $\widehat{\boldsymbol{x}}_{i+1} \leftarrow \text{F}(\boldsymbol{z}_i, \tau_i)$
  /* calculate a new estimate of variance $\boldsymbol{\tau}$ */
  $v_i \leftarrow \frac{\|\boldsymbol{r}_i\|_2^2 - M_R \sigma^2}{tr_G}$
  $\tau_i \leftarrow \boldsymbol{\nu}_i^1 \times \frac{tr_B \times v_i}{2M_T} + \boldsymbol{\nu}_i^2 \times \frac{\sigma^2 \times tr_W}{4M_T}$
end

Table 4.3: Assignment of learnable parameters to implement various detector types.

Detector↓ | Param.→	$\boldsymbol{W}$	$\boldsymbol{\Lambda}$	$\boldsymbol{\nu}^1$	$\boldsymbol{\nu}^2$
MMNet i.i.d [26]	$\boldsymbol{H}^T$	learned	learned	$\boldsymbol{\nu}^1 = \boldsymbol{\nu}^2$
MMNet [26]	learned, $\boldsymbol{\Theta}$	1	learned	$\boldsymbol{\nu}^1 = \boldsymbol{\nu}^2$
OAMPNet-H [31]	$\boldsymbol{H}^T$	learned	1	learned
OAMPNet-Hinv [31]	pinv($\boldsymbol{H}$)	learned	1	learned
OAMP [120]	$\boldsymbol{H}^T$	1	1	1

4.3.1 Computational complexity

The computational complexity of Algorithm 5 for one iteration without considering preprocessing is shown in Table 4.4. Preprocessing has to be done only once per channel realisation; therefore is not considered for now. Regardless of the configuration of W_{type}, the computational complexity is always the same. The computation is counted even if some learnable parameters are assigned to unity. This is required for implementing a single accelerator supporting multiple configurations. For calculating dot products, as needed in matrix-vector or vector-vector multiplications, a total number of additions equal to the number of elements in the dot product is used. From a computational point of view, this is equal to initialising the result variable with zero first and then adding products to it. This makes the notation of computational complexity simpler. By calculating the reciprocal of τ once, the number of divisions in Line 11 can be kept at one per iteration. Similarly, as the inverse of the column-wise sum is calculated once (Algorithm 3, Line 8) only M_T divisions are required instead of $M_T M_C$. The computational complexity for the F function is shown in Table 4.5.

Some example configurations and the corresponding computational complexity are shown in Table 4.4. In example 1, a configuration of 256x64_QAM64 is shown to require ≈ 35.7 k multiplications,≈ 6.2 k additions, and 67 divisions per transmit vector and iteration. Considering an LTE-A setup with 1200 SCs and a SCS of 15 kHz, the number of multiplications required becomes $35654 \times 15000 \times 1200 \approx 641$ billion. This clearly shows the need for custom hardware to accelerate the detection process.

Table 4.4: Computational complexity of Algorithm 5 for one iteration.

Line	Required Operations			Example 1			Example 2		
				$M_R = 256$	$M_T = 64$	$M_C = 8$	$M_R = 256$	$M_T = 128$	$M_C = 8$
	Mul	Add	Div	Mul	Add	Div	Mul	Add	Div
9	$M_R M_T$	$M_T M_T + M_R$		16384	4352		32768	16640	
10	$M_R M_T + M_T$	M_T		16448	64		32896	128	
11	C_{FM}	C_{FA}	C_{FD}	2560	1536	64	5120	3072	128
12	$M_R + 1$	$M_R + 1$	1	257	257	1	257	257	1
13	4	1	2	5	1	2	5	1	2
Sum of operations per iteration:				**35654**	**6210**	**67**	**71046**	**20098**	**131**

Table 4.5: Computational Complexity of the F function and G function as seen in Algorithm 3

Line	F Function			G Function (in addition to F)	
	C_{FM}	C_{FA}	C_{FD}	C_{GM}	C_{GA}
3	$2M_T M_C$	$M_T M_C$			
8		$M_T M_C$	M_T		
10	$M_T M_C$				
13	$M_T M_C$	$M_T M_C$		$M_T M_C + M_T$	$M_T M_C + M_T$
Total	$\boldsymbol{5M_T M_C}$	$\boldsymbol{3M_T M_C}$		$\boldsymbol{M_T M_C + M_T}$	$\boldsymbol{M_T M_C + M_T}$

4.4 Hardware Architecture

Based on the above discussion of Algorithm 5, this section introduces an FPGA hardware accelerator capable of real-time processing. The accelerator architecture is illustrated in Fig. 4.3 and solely considers the detector loop itself, not the preprocessing. This is because the preprocessing has to be performed only once per channel coherence interval. The inputs $1/tr_G$, tr_{Wi}, tr_{Bi} and σ^2 are bundled in an input named "channel statistics / ChStat" and provided to the accelerator. On a similar principle as mentioned above, the inverse $1/tr_G$ instead of tr_G is supplied as an accelerator input. This saves many divisions as the inverse only needs to be calculated once per channel coherence interval, not each in each iteration of each channel use. In some detector cases such as MMNet, the inputs of tr_{Wi} and tr_{Bi} should be calculated separately for each iteration, forming vectors of length M_T. For simplicity reasons, the below-presented accelerator has these inputs implemented as scalars. Implementing them as vectors is easily possible and will not change the results significantly.

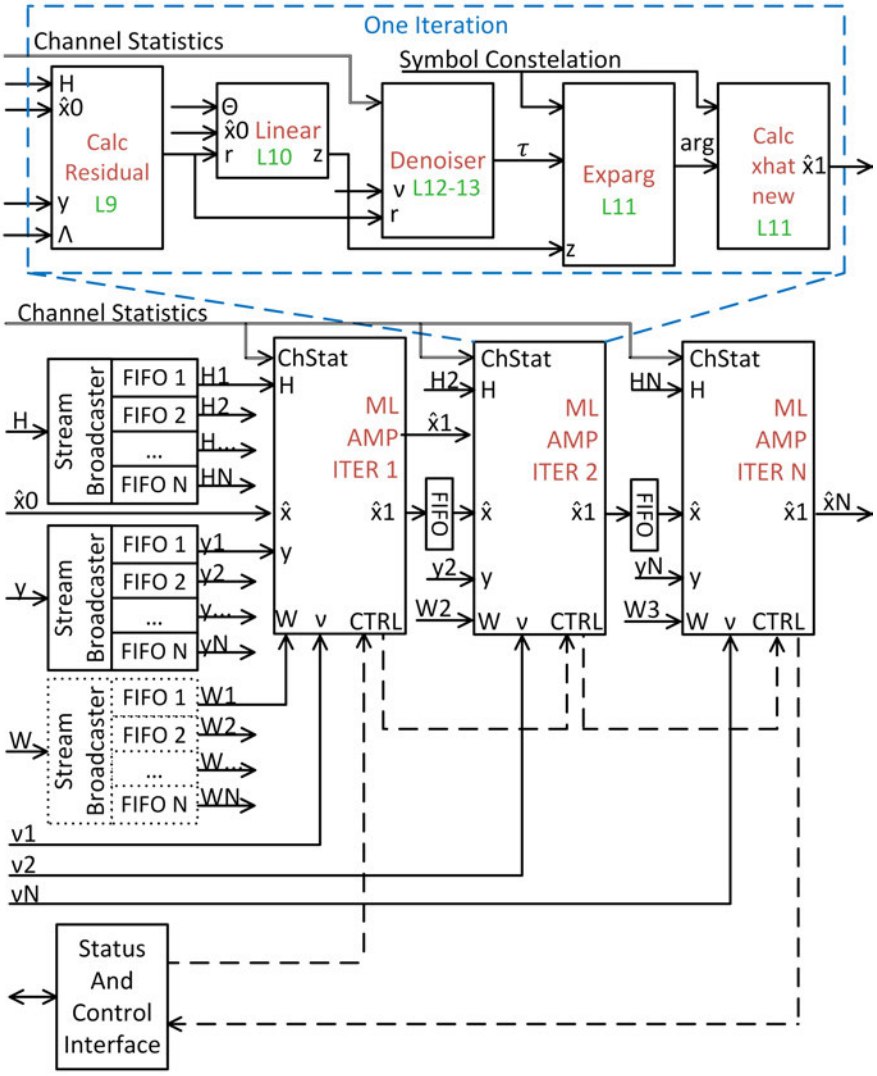


Figure 4.3: Deep Pipelined, machine learning extended OAMP accelerator. The green numbers are the line numbers in Algorithm 5.

The learnable parameters $\boldsymbol{\Theta}$, $\boldsymbol{\Lambda}$, $\boldsymbol{\nu}^1$ and, $\boldsymbol{\nu}^2$ are provided as inputs to the accelerator as well. In MMNet, $\boldsymbol{\nu}^1$ equals $\boldsymbol{\nu}^2$, and both are vectors of length M_T instead of scalars. This is to scale the noise variance of each symbol individually. The current implementation keeps these parameters as scalars per iteration. The functionality for each iteration is divided into multiple functional blocks, which form a pipeline for maximum throughput. The design of the OAMP iteration block is done in Xilinx's Vitis HLS tool.

The first step in each algorithm iteration is calculating the **residual** ($\boldsymbol{r}_i$). The channel matrix $\boldsymbol{H}$ is streamed via a 32-word wide AXI-stream. Then the **noisy, intermediate signal** $z \in \mathbb{R}^{M_T}$ is calculated from the streamed $\boldsymbol{W}_i$ matrix, from Λ_i, and the residual. Two potentially learnable parameters regulate the generation of z: the parameter $\boldsymbol{\Lambda}_i$ and the Matrix $\boldsymbol{W}_i$. Next, the **error variance** v_i is calculated. This can be done before or after the denoising step. Since the error variance should not be exactly zero, it is set to the LSB if the data-type resolution is not sufficient:

$$v_i = \max\left(\frac{\|\boldsymbol{r}_i\|_2^2 - M_R\sigma^2}{tr_G}, LSB\right) \tag{4.27}$$

The calculation of the **noise variance** τ involves two learnable, scalable parameters ν^1 and ν^2 as shown in Algorithm 5, Line 13. The next step is Gaussian **denoising** by element-wise thresholding as shown in Algorithm 3. First, the argument of the exponential is calculated for each combination of z and all possible symbol values $\boldsymbol{c}$, leading to $\boldsymbol{arg} \in \mathbb{R}^{M_T, M_C}$. This corresponds to the fourth block in the accelerator in Fig. 4.3. Next, the exponential of each element of $\boldsymbol{arg}$ is taken by looking up the value of the exp-function across its input range. In this design, the LUT contains 1024 entries. The entries of $\boldsymbol{arg}$ are summed across the constellation points for each symbol to form $\boldsymbol{ssum} \in \mathbb{R}^{M_T}$ (Algorithm 3). Finally, the elements of $\boldsymbol{arg}$ are scaled by $\boldsymbol{ssum}$ and multiplied with the constellation points to form the output of the iteration, namely the new estimate of $\widehat{\boldsymbol{x}}$.

Multiple iterations and supporting blocks are integrated at RTL level. M_R, M_T, and the modulation type are compile-time parameters. For each combination of parameters, a new high-level synthesis run is performed. The larger inputs to the accelerator ($\boldsymbol{H}, \boldsymbol{W}, \boldsymbol{\Lambda}, \widehat{\boldsymbol{x}}_0, \boldsymbol{y}$, and $\widehat{\boldsymbol{x}}_1$) are implemented as AXI-streams [139]. This simplifies the integration of multiple iterations. Buffers have to be introduced for the variables shared across multiple iterations since each iteration requires the streamed data at a different time. These buffers consist of stream broadcasters and FIFOs. The maximum stream width is limited to 512 bits per clock cycle for the large matrices $\boldsymbol{H}$ and $\boldsymbol{W}$. In some use cases such as MMNet, each iteration requires its distinct, learned matrix ($\boldsymbol{W}_i = \boldsymbol{\Theta}_i;\ \forall i \in [0, M_I]$). In this case, the buffer on $\boldsymbol{W}$ might be removed and replaced by multiple, distinctive inputs of $\boldsymbol{W}$ for each iteration. This slightly decreases the required resources and does not affect accelerator performance. Scalar inputs such as the channel statistics and ν are implemented via direct memory access. An overall status and control interface is implemented to control the accelerator. All data inputs and outputs are quantised with 25-bit word length and 15-bit fractional length. Most internal calculation results also have this quantisation; only $\boldsymbol{W}$ and $\boldsymbol{H}$ have a word length of 16 bits, of which are ten fractional bits. This still achieves good accuracy while saving memory and streaming time on these large matrices.

4.5 Results and Performance

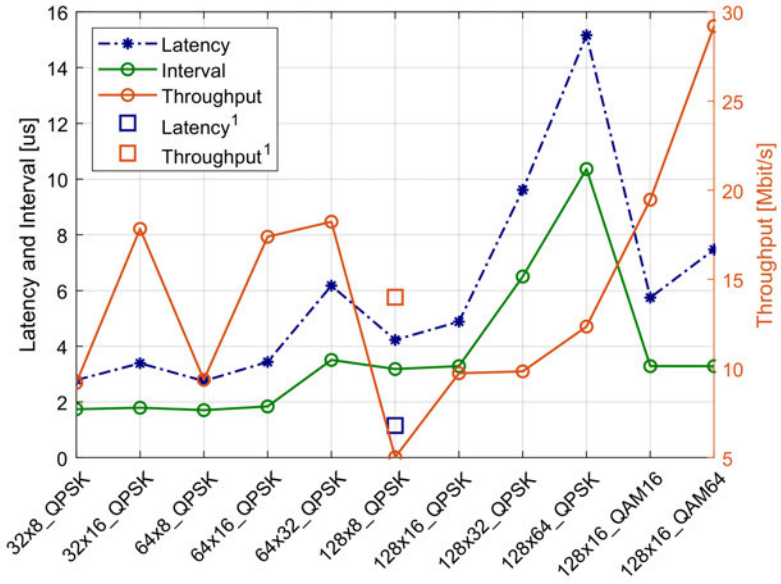
The hardware architecture is evaluated for a Xilinx Zynq Ultrascale+ XCZU27DR-2 FPGA [7]. After HLS, RTL synthesis is performed in an Out Of Context (OOC) run, and the reported clock frequency and utilisation results are obtained post Place & Route (P&R). The design is tested by exporting the required detector inputs and outputs such as $\boldsymbol{H}$, $\boldsymbol{y}$, $\boldsymbol{x}$, $\widehat{\boldsymbol{x}}$, and more to files. During high-level C simulation and RTL verification, these inputs are read in the testbench and provided to the Device Under Test (DUT). Then, the hardware produced $\widehat{\boldsymbol{x}}$ variable is compared with the expected $\widehat{\boldsymbol{x}}$ variable produced by Tensorflow simulations.

Fig. 4.4a and Fig. 4.4b show latency, interval, throughput, and utilisation for multiple configurations of the core algorithm for one iteration as displayed in Fig. 4.3. The latency is measured from when the start signal is set high until the first result is fully streamed out. The interval describes the time from when one output has been produced fully until the next (i.e. the time between when different $\widehat{\boldsymbol{x}}$ are made available). This time assumes the pipeline is filled and can also be called the pipelined interval. The throughput on the right y-axis describes the total throughput of the detector considering all users and is given in Mbit/s:

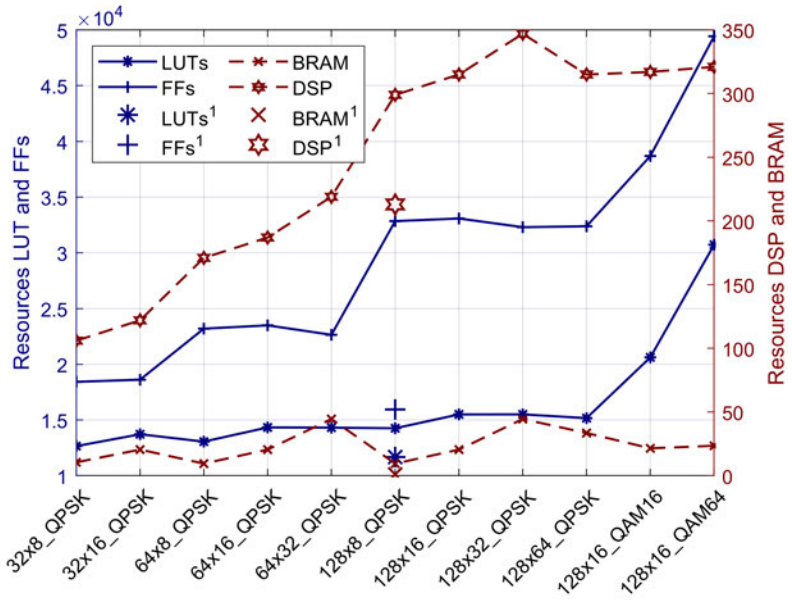
$$Throughput\ [\text{Mbit/s}] = \frac{\log_2(|\mathcal{C}_\mathcal{C}|)\ [\text{bit}] \times M_T}{Interval\ [\mu\text{s}]}. \tag{4.28}$$

The influence of the antenna configuration on the detector latency and interval can be seen in Fig. 4.4a. The larger the number of base-station antennas becomes ($\overline{M_R}$), the more influence the number of connected users ($\overline{M_T}$) has on the latency and interval. For example, the difference in latency between 64x16_QPSK and 64x32_QPSK is $2.74\,\mu$s, while the difference in latency between 128x16_QPSK and 128x32_QPSK is already $4.73\,\mu$s. The interval peaks at the configuration of 128x64_QPSK with $10.36\,\mu$s. The modulation type only has a small influence on latency and no influence on interval and throughput. This property is good for high throughput use cases with high order modulation schemes such as for eMBB.

The number of DSP and BRAM resources is not influenced significantly by the modulation scheme; however, the number of LUT and FF CLB slices increases. The increase from 128x64_QPSK to 128x64_QAM64 is $\approx$17 kFF and $\approx$16 kLUT. Considering the number of these resources in modern FPGAs, the increase is negligible. The targeted FPGA has 4272 DSP units; however, the usage of up to 347 DSPs per iteration will be the resource limiting factor in implementing the detector. The BRAM usage is between 9.5 and 44.5 BRAMs per iteration and increases only slowly with the number of users. The number of base-station antennas seems not to influence memory usage. This is because one BRAM can only provide a maximum of 36 bits in parallel. The more users



(a) Latency, interval, and throughput of one instance



(b) Resource utilization of one instance

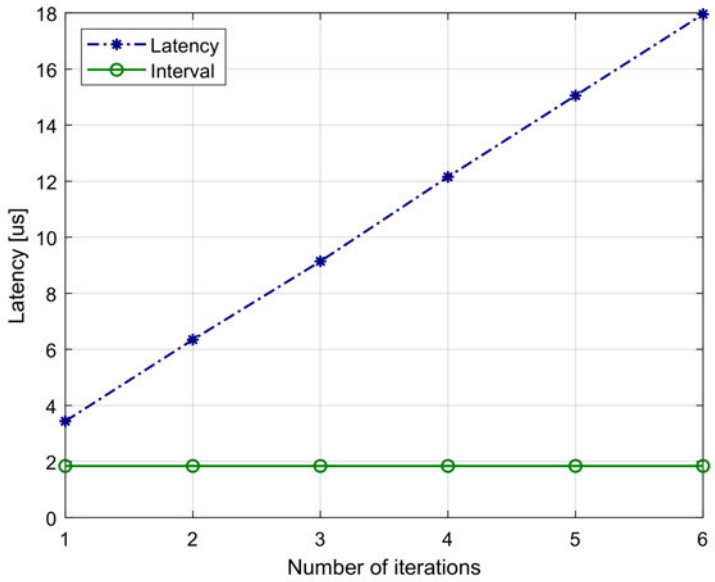
Figure 4.4: Latency, interval, throughput, and resource requirements for various complex system configurations $\overline{M_R}$x$\overline{M_T}$ and modulation types QPSK, QAM16, and QAM64. [1]Reported Results from [4].

are connected, the more data needs to be processed in parallel to keep the latency low. This mandates higher BRAM usage. However, the depth of the BRAM blocks is sufficient to store larger configurations of $\boldsymbol{H}$ and $\boldsymbol{W}$ without using more BRAM.

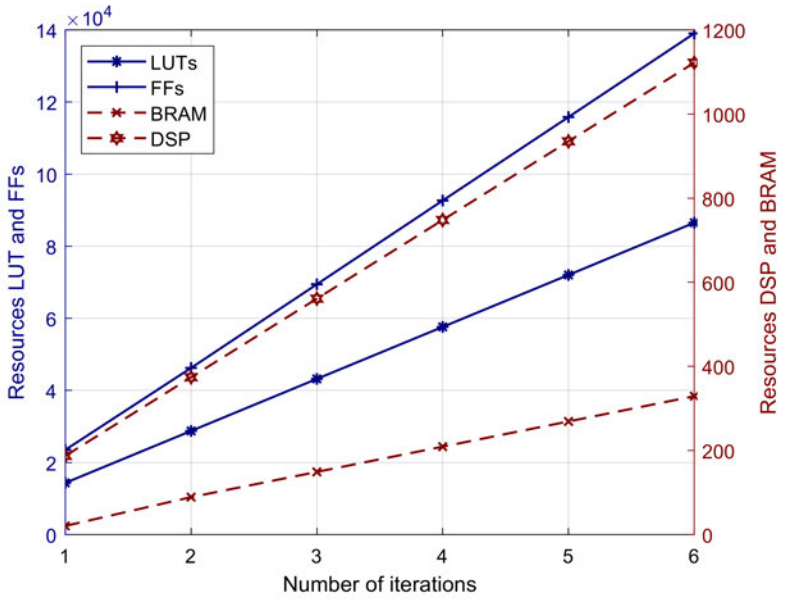
Comparing the AMP detector in [4] with the configuration 128x8_QPSK, it can be seen that the throughput is lower and the latency higher in the proposed accelerator. Also, the resource usage is higher, and the reported numbers in [4] are for three iterations compared to the one iteration of the proposed accelerator. One of the decisive differences is that the proposed accelerator is based on OAMP, whereas the work in [4] implements the AMP algorithm which is less stable for non-i.i.d. Gaussian entries of $\boldsymbol{H}$. This is one of the motivations for developing the new machine learning enhanced detector named LAMANet in Chapter 5. Of course, the other difference is that the proposed accelerator supports machine learning enhancements, which require a full channel matrix instead of the Gram channel matrix in the linear part ($\boldsymbol{G}_T = \boldsymbol{H}^T\boldsymbol{H}$). An ASIC implementation of the LAMA algorithm can also be found in [140] for reference.

Even though one iteration could be repeatedly invoked, multiple iterations can be pipelined in a deeply pipelined design for maximum throughput. As mentioned above, supporting blocks like stream broadcasters and FIFOs are required to allow each iteration to stream the required data when needed. Further, the resulting $\widehat{\boldsymbol{x}}$ values must be buffered between iterations. The results of an integration of multiple iterations are shown for an antenna configuration of 128x32_QPSK in Figs. 4.5a and 4.5b and Table 4.6.

As expected, the latency increases linearly with the number of iterations integrated. The Interval stays constant at $\approx$2 µs. The number of resources increases linearly with the number of integrated iterations as well. Most critically, the number of DSP resources is large, and improvements are highly desired.



(a) Initial and pipelined latency multiple iterations integrated



(b) Resource utilization multiple instances integrated

Figure 4.5: Latency, interval, throughput, and resource requirements when integrating multiple iterations for an antenna configuration of 128x32_QPSK.

Table 4.6: Latency and resource usage for multiple integrated iterations in the configuration 128x32QPSK.

	Integrated Iterations					
	1	2	3	4	5	6
Latency [us]	3.437	6.346	9.14	12.151	15.053	17.955
Interval [us]	1.84	1.84	1.84	1.84	1.84	1.84
LUT	14332	28742	43182	57606	72020	86504
FF	23490	46226	69438	92664	115827	139036
BRAM total	20.5	89	149	209	269	329
DSP total	187	374	561	748	935	1122

4.6 Summary

Based on the previous work on ML-enhanced OAMP detection, this chapter presented an OAMP algorithm with multiple learnable parameters, such that the algorithm can accommodate the previously proposed MMNet and OAMP-Net detectors. A computational complexity analysis highlights the challenge of real-time processing, and a custom FPGA accelerator is proposed.

The FPGA-detectors performance is evaluated across multiple configurations and shows a pipelined latency between $\approx 2\,\mu s$ and $\approx 10\,\mu s$ depending on the configuration. However, the resource usage of DSP blocks is high compared to previous designs without machine learning, and the throughput is considerably lower.

The work of this chapter functions as a baseline for the following Chapter 5, and the shortcomings of this accelerator motivate a deeper investigation of OAMP based algorithms. The resource usage needs to be reduced (in particular the DSP usage), and some more fundamental issues, such as the online training requirement for MMNet and the excessive computational complexity in OAMP-Net, need to be addressed.

Chapter 5

LAMANet: An Improved ML-enhanced AMP detector for Massive MIMO

5.1 Introduction

In Chapter 4 the OAMP-based, ML-enhanced detection algorithms MMNet and OAMP-Net have been analysed. Commonalities between them have been identified and a unified OAMP algorithm was proposed. Based on this a hardware architecture was developed and an FPGA accelerator was implemented. The profiling results highlighted performance issues in these detectors. When compared to similar, classical AMP algorithms, the resource usage is larger while throughput is lower. This is due to the overly complex computational structures of the OAMP algorithm, and of course due to the fact that ML-enhancements are supported. This is even without considering OAMP-Net proposals where the MMSE matrix has to be calculated in every iteration of OAMP-Net. Even in this case, the preprocessing in OAMP-Net is of significant complexity.

To reduce complexity, we propose a novel ML-enhanced detector algorithm based on AMP in Section 5.2. This algorithm is named LAMANet since AMP for MIMO detection was first proposed as LArge MIMO AMP (LAMA) [118, 119]. Due to the training of the learnable parameters, the instability in the AMP algorithm can be avoided and it will be shown to perform on the same level as MMNet while reducing complexity.

MMNet's performance on realistic channels is impressive, however, the cost for this performance is its requirement of online training. Online training means that the detector can only be used after training it on the specific realisation of the channel in the current coherence interval. The training could be performed based on on-the-fly generated, random input data and the current channel matrix. If the detector was trained on a diverging channel matrix, it suffers complete performance loss. In practice, this means an excessively high detection latency as the training of the detector needs to be completed first. To mitigate this issue, we propose to incorporate the learnable parameters in a novel way, allowing LAMANet to be initialised with the MMSE matrix (Section 5.2.2) or the OAMP-optimal matrix (Section 5.2.2).

Based on the LAMANet algorithm a deeply pipelined FPGA accelerator is proposed in Section 5.3. This accelerator is similar to the one proposed in Section 4.4, however, it is simplified due to the simpler algorithm and significant performance improvements are shown.

Link-level simulations with a realistic channel model are performed in Section 5.4. Multiple classical and ML-enhanced detectors are compared for various configurations. The results show that the detection performance of LAMANet is close to MMNet in terms of SER.

In Section 5.5 the FPGA accelerator designs for different versions of LAMANet are provided and compared to other conventional and ML-enhanced massive MIMO detection algorithms. The results show the competitive performance of LAMANet.

5.1.1 Contribution

The core contribution of this chapter is as follows. From an algorithmic point of view we

- proposes to base the detector on AMP instead of the more complex OAMP algorithm to reduce complexity,

- propose a new way of incorporating learnable matrices in the AMP algorithm to prevent loss of performance in untrained detectors, thereby relaxing the online training requirement of MMNet, and
- remove unnecessary computations in the AMP algorithm as their functionality is replaced by the learned parameters.

From the hardware perspective, a deeply-pipelined hardware accelerator is proposed and synthesised for the Xilinx RFSoC FPGA. The presented work is important as it demonstrates one of the first real-time machine-learning massive MIMO detectors.

LAMANet is designed such that it could be easily integrated into the receiver chain as seen in Fig. 2.5. Apart from the usual integration tasks such as matching the interfaces of the design, and providing configuration and control options, there are no special requirements when it comes to integration. This is an important advantage over designs that model and transform the entire reception chain into a neural network based system. Examples are autoencoder-based designs such as in [98]. Another commonality to classical detector algorithms is the need for generating some inputs to the detector once per channel realisation. This preprocessing could be done in the channel estimator or in a dedicated module in the integrated receiver. Since online learning is required to some extent for LAMANet, it is important to allow the learnable parameters to be easily updated during deployment. Apart from this, no special requirements for integration are identified.

The LAMANet algorithms, accelerator architectures and digital circuits are covered by the publication in [29]:

> S. Brennsteiner, T. Arslan, J. S. Thompson, and A. McCormick, “LAMANet: A Real-Time, Machine Learning-Enhanced Approximate Message Passing Detector for Massive MIMO,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, pp. 1–14, 2022

5.2 LAMANet Algorithm

For a more detailed algorithmic introduction than presented here, please refer to Section 4.2. The AMP- and OAMP-type of algorithms allow for iteratively solving the massive MIMO detection problem of Eq. (5.1).

$$\widehat{\boldsymbol{x}}_{ML} = \underset{\boldsymbol{x}\in\mathcal{C}_R^{M_T}}{\arg\min} \ \|\boldsymbol{y} - \boldsymbol{H}\boldsymbol{x}\|_2^2 \tag{5.1}$$

The detectors consist of a linear part (Eq. (5.2) and Eq. (5.3)) and a denoiser part (Eq. (5.4)).

$$\boldsymbol{r}_k = \boldsymbol{y} - \boldsymbol{H}\widehat{\boldsymbol{x}}_k + \boldsymbol{v}_k, \tag{5.2}$$

$$\boldsymbol{z}_k = \boldsymbol{H}^T\boldsymbol{r}_k + \widehat{\boldsymbol{x}}_k, \tag{5.3}$$

$$\widehat{\boldsymbol{x}}_{k+1} = \eta(\boldsymbol{z}_k), \tag{5.4}$$

where $\eta(\cdot)$ is the denoiser function. Following this, the LAMA algorithm can be derived as seen in Algorithm 6.

5.2.1 LAMANet

OAMP was introduced to stabilize the AMP algorithm in case the entries of the transformation matrix are not strictly adhering to an i.i.d. Gaussian distribution. This makes OAMP applicable to a wider range of problems including detection in massive MIMO. TISTA[122], OAMPNet[31], MMNetiid, and MMNet[26] are based on this method, as introduced in Section 4.2.3. However, the reason why MMNet performs particularly well on real-world channels far from the ideal i.i.d. case is that in the linear function, the noise present in $\boldsymbol{z}$ can be shaped very close to a Gaussian distribution by learning the matrices $\boldsymbol{W}_i$.

Algorithm 6: LAMA algorithm [117].

input : Received signal $\boldsymbol{y} \in \mathbb{R}^{M_R}$
input : Channel matrix $\boldsymbol{H} \in \mathbb{R}^{M_R \times M_T}$
input : Noise variance $\sigma^2 \in \mathbb{R}$
output: $\widehat{\boldsymbol{x}}_{I+1} \in \mathbb{R}^{M_T}$

/* initialisation */
1 $\widehat{\boldsymbol{x}}_1 \in \mathbb{R}^{M_T} \leftarrow \mathbb{E}_X[\mathcal{C}_\mathcal{R}]$
/* preprocessing */
2 $\beta \leftarrow M_T/M_R;\ \boldsymbol{y}_{MRC} \leftarrow \boldsymbol{H}^T\boldsymbol{y};\ \boldsymbol{G}_T \leftarrow \boldsymbol{H}^T\boldsymbol{H}$
/* iteration loop */
3 **for** $i \leftarrow 1$ **to** M_I **do**
4 $\quad \boldsymbol{z}_i \leftarrow \boldsymbol{y}_{MRC} + (\boldsymbol{I_T} - \boldsymbol{G_T})\widehat{\boldsymbol{x}}_i + \boldsymbol{v}_i$ // linear function
5 $\quad \widehat{\boldsymbol{x}}_{i+1} \leftarrow \mathrm{F}(\boldsymbol{z}_i, \sigma^2/2 + \beta\tau_i)$ // calculate estimate of $\widehat{\boldsymbol{x}}$
6 $\quad \boldsymbol{\phi}_{i+1} \leftarrow \mathrm{G}(\boldsymbol{z}_i, \sigma^2/2 + \beta\tau_i)$ // calculate estimate of variance $\boldsymbol{\tau}$
7 $\quad \tau_{i+1} \leftarrow \sum_{k=1}^{M_T} \boldsymbol{\phi}_{i,k}$
8 $\quad \boldsymbol{v}_{i+1} \leftarrow \frac{\beta\tau_{i+1}}{\sigma^2/2+\beta\tau_i}(\boldsymbol{z}_i - \widehat{\boldsymbol{x}}_i)$ // calculate new onsager term
9 **end**

In the results below, we show that based on this reasoning, the usage of the complex OAMP algorithm is not required for machine learning-enhanced massive MIMO detection. Instead, the more traditional, and computationally simpler AMP algorithm can be enhanced with learnable parameters.

Therefore, we propose to add similar learnable parameters to the LAMA algorithm instead of to the OAMP algorithm. The modifications themselves follow closely the proposal made in MMNet. The simple case where the entries in $\boldsymbol{H}$ are i.i.d are not considered as the channel model is important to judge the performance of the algorithm. We call this first proposal LAMANetBL for LAMANetBaseLine, as it closely follows MMNet's extensions for the LAMA algorithm and since we will use it for comparison. The addition of the learnable parameter $\boldsymbol{\Theta}$ to the expanded, linear part of Algorithm 6, Line 4 is as follows:

$$\boldsymbol{z}_i = \widehat{\boldsymbol{x}}_i + \boldsymbol{\Theta}_i(\boldsymbol{y} - \boldsymbol{H}\widehat{\boldsymbol{x}}_i) + \boldsymbol{v}_i; \quad \boldsymbol{\Theta}_i \text{ are learned.} \tag{5.5}$$

Similarly, a noise scaling parameter is introduced to Algorithm 6, Line 7 as follows:

$$\boldsymbol{\tau}_i = \boldsymbol{\theta}_i \sum_{k=1}^{M_T} \boldsymbol{\phi_{i,k}}; \quad \boldsymbol{\theta}_i \in \mathbb{R}^{M_T} \text{ are learned.} \tag{5.6}$$

In LAMANetBL the learnable matrix $\boldsymbol{\Theta}_i$ has the form of $\mathbb{R}^{M_T,M_R}$. The noise variance $\boldsymbol{\tau}_i$ is now a vector of length M_T instead of a scalar. This allows to scale the noise variance individually for each connected UEs.

To reduce computational complexity we consider the condensed form of the linear part as shown in Algorithm 6, Line 4 and enhance it with two learnable matrices per iteration:

$$\boldsymbol{z}_i = \boldsymbol{y_{MRC}} + (\boldsymbol{I_R} - (\boldsymbol{\Theta}_{i,1} \odot \boldsymbol{G}) \oplus \boldsymbol{\Theta}_{i,2})\widehat{\boldsymbol{x}}_i + \boldsymbol{v}_i; \tag{5.7}$$

$$\text{learned parameter: } \boldsymbol{\Theta} \in \mathbb{R}^{M_I \times 2 \times M_T \times M_T}. \tag{5.8}$$

The learnable matrices are element-wise multiplied and added respectively. We name this approach LAMANetC (for LAMANet Condensed). The number of multiplications in the linear part for LAMANet is $2M_RM_T$ whereas for LAMANetC it is $2M_TM_T$. In the large system limit ($M_T << M_R$), this difference can become significant. In the condensed form it is also possible to define a learnable matrix with the dimensions $M_T \times M_T$ and multiply it with $\boldsymbol{G}$ directly, however; this would be more expensive in terms of computational cost with $(M_T)^3$ multiplications. It is important to notice that the above simplification of the linear function can only be applied in AMP-type detectors such as LAMA and LAMANet, not in OAMP-type detectors such as in OAMPNet and MMNet. The reason is that in the latter, the residual $\boldsymbol{r}_i$ has to be generated separately as it is used in the estimate of the noise variance (Algorithm 4, Line 11).

To confirm the validity of element-wise operations, we propose and evaluate another design, which defines the linear function as follows:

$$\boldsymbol{z}_i = \widehat{\boldsymbol{x}}_i + \boldsymbol{H}^T(\boldsymbol{y} - ((\boldsymbol{\Theta}_{i,1} \odot \boldsymbol{H}) \oplus \boldsymbol{\Theta}_{i,2})\widehat{\boldsymbol{x}}_i) + \boldsymbol{v}_i; \tag{5.9}$$

$$\text{learned parameter: } \boldsymbol{\Theta} \in \mathbb{R}^{M_I \times 2 \times M_R \times M_T}. \tag{5.10}$$

We call this approach LAMANetEW (for LAMANet Element-Wise). Its computational complexity is higher than LAMANet, as additional $M_R \times M_T$ multiplications and additions are required.

The results in Section 5.4 show that it is unnecessary to calculate a new noise estimate($\boldsymbol{\tau}_{i+1}$) and Onsager term ($\boldsymbol{v}_i$) in every iteration when online learning on a specific channel is performed. This is because the noise variance can be learned by $\boldsymbol{\theta}$ for each layer and the Onsager term can be learned by $\boldsymbol{\Theta}$ in the linear part. This saves computational cost in the LAMANet-type algorithms as lines 6-8 of Algorithm 6 can be omitted. We refer to this detector type as LAMANet.

5.2.2 LAMANetMMSE

As mentioned above, MMNet, LAMANetBL, LAMANetC, and LAMANet perform poorly when not trained on a specific channel realisation. On the other hand, OAMPNet performs decently well without online training due to the choice of the optimal matrix in its linear part (Algorithm 4, Line 12 to Line 20, $W_{type} = opt$). However, its computational complexity can be infeasibly high as it requires matrix inversion in every iteration of the algorithm. Also, its trained performance is lower than that of MMNet. We propose to initialize LAMANet with the MMSE matrix in the linear part which avoids both problems. The MMSE matrix is calculated only once per coherence interval, thereby reducing the computational effort to acceptable levels. This is while the detection performance is also kept at reasonable levels when not trained on a specific channel - relaxing the online training requirement. We name this approach LAMANetMMSE and show it in Algorithm 7. In this algorithm, the element-wise modification via learnable matrices of $\boldsymbol{W}_i$ is required.

Algorithm 7: LAMANetMMSE algorithm. Parameters with light-green background colour are learnable.

input : Received signal $\boldsymbol{y} \in \mathbb{R}^{M_R}$
input : Channel matrix $\boldsymbol{H} \in \mathbb{R}^{M_R \times M_T}$
input : Noise variance $\sigma^2 \in \mathbb{R}$
input : learned $\boldsymbol{\Theta} \in \mathbb{R}^{M_I \times 2 \times M_T \times M_R}$; $\boldsymbol{\theta} \in \mathbb{R}^{M_I \times M_T}$
output: $\widehat{\boldsymbol{x}}_{M_I+1} \in \mathbb{R}^{M_T}$

initialisation: $\widehat{\boldsymbol{x}}_1 \in \mathbb{R}^{M_T} \leftarrow \mathbb{E}_X[X]$
/* preprocessing */
$\boldsymbol{G} \leftarrow \boldsymbol{H}^T\boldsymbol{H}$
$\boldsymbol{W}_i \leftarrow \boldsymbol{H}(\boldsymbol{G} + \sigma^2\boldsymbol{I_T})^{-1}$ // MMSE matrix calculation
/* detector loop */
for $i \leftarrow 1$ **to** M_I **do**
  /* linear function */
  $\boldsymbol{r}_i \leftarrow \boldsymbol{y} - \boldsymbol{H}\widehat{\boldsymbol{x}}_{i+1}$
  $\boldsymbol{z}_i \leftarrow \widehat{\boldsymbol{x}}_i + ((\boldsymbol{\Theta}_{i,1} \odot \boldsymbol{W}_i) \oplus \boldsymbol{\Theta}_{i,2})\boldsymbol{r}_i$
  /* calculate new estimate of $\widehat{\boldsymbol{x}}$ */
  $\boldsymbol{\tau} \leftarrow \boldsymbol{\theta}_i / \sigma^2$
  $\widehat{\boldsymbol{x}}_{i+1} \leftarrow \mathrm{F}(\boldsymbol{z}_i, \boldsymbol{\tau})$ // denoising function
end

5.2.3 LAMANetOpt

In a similar fashion to LAMANetMMSE, and as proposed in OAMPNet, the matrix $\boldsymbol{W}_i$ can be set to the optimal matrix [31]:

$$\boldsymbol{W}_i = \frac{2M_T\hat{\boldsymbol{W}}_{\boldsymbol{i}}}{\mathrm{tr}(\hat{\boldsymbol{W}}_{\boldsymbol{i}}\boldsymbol{H})}, \tag{5.11}$$

with:

$$\hat{\boldsymbol{W}}_{\boldsymbol{i}} = v_i^2\boldsymbol{H}(v_i^2\boldsymbol{H}^T\boldsymbol{H} + \frac{\sigma^2}{2}\boldsymbol{I_T})^{-1}, \tag{5.12}$$

$$v_i^2 = \frac{\|\boldsymbol{r}_i\|_2^2 - M_R\sigma^2}{\mathrm{tr}(\boldsymbol{H}^T\boldsymbol{H})}. \tag{5.13}$$

For the stability of the algorithm, it is important that v_i^2 does not become zero or negative [120]. For this reason, we set $v_i^2 = \max(v_i^2, LSB)$, where LSB is the least significant bit of the fixed-point datatype of v_i^2.

Since $\boldsymbol{W}_i$ ultimately depends on the residual $\boldsymbol{r}_i$, it has to be calculated in every iteration of the algorithm. This is undesirable because the matrix operations are expensive - in particular the matrix inverse. To reduce the complexity we propose to use the eigenvalue decomposition of the MMSE matrix in the pre-processing, i.e. once per channel coherence interval:

$$[\boldsymbol{V}\ \boldsymbol{D}] = \mathrm{eig}(\boldsymbol{H}^T\boldsymbol{H}). \tag{5.14}$$

Where $\boldsymbol{V} \in \mathbb{R}^{M_T \times M_T}$ is a diagonal matrix with the eigenvalues on its diagonal and $\boldsymbol{D} \in \mathbb{R}^{M_T \times M_T}$ holds in its columns the right eigenvectors. Then, the inverted MMSE matrix can be calculated in each iteration as:

$$\hat{\boldsymbol{W}}_{\boldsymbol{i}} = v_i^2 \boldsymbol{H}(\boldsymbol{V}\frac{\boldsymbol{I_T}}{\boldsymbol{M}_i}\boldsymbol{V}^T), \tag{5.15}$$

with the argument of the inverse as:

$$\boldsymbol{M}_i = \boldsymbol{D}v_i^2 + \frac{\sigma^2}{2}\boldsymbol{I_T}. \tag{5.16}$$

This proposal is named LAMANetOpt and the complete algorithm is shown in Algorithm 8.

5.2.4 Other Initialisations

A low-complexity initialisation of the LAMANet detector can be given by setting $\boldsymbol{W}_i = \boldsymbol{H}^T,\ i \in [1, M_I]$, or by setting $\boldsymbol{W}_i = \mathrm{diag}(\boldsymbol{H}^T),\ i \in [1, M_I]$, where the function $\mathrm{diag}()$ takes the diagonal of an input matrix. Both schemes do not require any additional computations per coherence interval, however, they reduce the untrained performance. For this reason, we do not further evaluate them.

Algorithm 8: LAMANetOpt algorithm. Parameters with light-green background colour are learnable.

input : Received signal $\boldsymbol{y} \in \mathbb{R}^{M_R}$
input : Channel matrix $\boldsymbol{H} \in \mathbb{R}^{M_R \times M_T}$
input : Noise variance $\sigma^2 \in \mathbb{R}$
input : learned $\boldsymbol{\Theta} \in \mathbb{R}^{M_I \times 2 \times M_T \times M_R}$; $\boldsymbol{\theta} \in \mathbb{R}^{M_I \times M_T}$
output: $\widehat{\boldsymbol{x}}_{M_I+1} \in \mathbb{R}^{M_T}$

initialisation: $\widehat{\boldsymbol{x}}_1 \in \mathbb{R}^{M_T} \leftarrow \mathbb{E}_X[X]$
`/* preprocessing */`
$\boldsymbol{G} \leftarrow \boldsymbol{H}^T\boldsymbol{H}; \quad tr_G = \mathrm{tr}(\boldsymbol{G})$
$[\boldsymbol{V}\ \boldsymbol{D}] = \mathrm{eig}(\boldsymbol{G})$ `// eigen decomposition`
`/* detector loop */`
for $i \leftarrow 1$ **to** M_I **do**
  `/* linear function */`
  $\boldsymbol{r}_i \leftarrow \boldsymbol{y} - \boldsymbol{H}\widehat{\boldsymbol{x}}_{i+1}$
  $v_i^2 = (\|\boldsymbol{r}_i\|_2^2 - M_R\sigma^2)/(tr_G)$
  $\boldsymbol{M}_i = \boldsymbol{D}v_i^2 + \sigma^2\boldsymbol{I_T}$
  $\hat{\boldsymbol{W}}_{\boldsymbol{i}} = v_i^2\boldsymbol{H}(\boldsymbol{V}\frac{\boldsymbol{I_T}}{\boldsymbol{M}_i}\boldsymbol{V}^T)$
  $\boldsymbol{W}_i = 2M_T\hat{\boldsymbol{W}}_{\boldsymbol{i}}/\mathrm{tr}(\hat{\boldsymbol{W}}_{\boldsymbol{i}}\boldsymbol{H})$
  $\boldsymbol{z}_i \leftarrow \widehat{\boldsymbol{x}}_i + ((\boldsymbol{\Theta}_{i,1} \odot \boldsymbol{W}_i) \oplus \boldsymbol{\Theta}_{i,2})\boldsymbol{r}_i$
  `/* calculate new estimate of` $\widehat{\boldsymbol{x}}$ `*/`
  $\boldsymbol{\tau} \leftarrow \boldsymbol{\theta}_i/\sigma^2$
  $\widehat{\boldsymbol{x}}_{i+1} \leftarrow \mathrm{F}(\boldsymbol{z}_i, \boldsymbol{\tau})$ `// denoising function`
end

The initialisation of the weights in HyperMIMO is provided by meta-learning [124]. A dedicated neural network learns the weights and provides them to an MMNet-type detector. The neural network produces the weights based on QR-decomposed channel matrices as its input and can accommodate slow changes in the user location. However, initial training on the user positions is required and the additional neural network and QR-decomposition are expensive. We see HyperMIMO as an alternative to the MMSE and Opt initialisation.

5.2.5 Algorithmic Complexity

Algorithmic complexity is one of the main performance metrics to evaluate for any massive MIMO detector, as it will have a major impact on circuit performance such as latency, resource requirements, and throughput. In Table 5.1, the number of multiplications and additions for various detector variants is listed based on Algorithms 4, 6 and 7. The first column indicates the line number of the corresponding algorithms. The sum of required operations for preprocessing (i.e. once per channel interval) and the sum of required operations performed per transmitted vector $\boldsymbol{x}$ (i.e. once per channel use) are shown in the last two rows. The green background color indicates the former, while light gray indicates the latter. The complexity of the denoiser functions F and G is listed in Table 5.2. Required divisions are not listed, however, these can be easily estimated from the algorithmic listings. The complexity for various antenna configurations and modulation types is shown in Fig. 5.1. For clarity, only the number of multiplications is shown. The number of additions is approximately equal to the number of multiplications.

The advantage of using the AMP-based LAMANet over the OAMP-based OAMPNet or MMNet can be easily seen in Fig. 5.1a. OAMP-based designs need a large amount of preprocessing, while the amount of preprocessing in the AMP-based designs is zero (not shown on the log scale). For larger system configurations the preprocessing effort in OAMPNet and MMNet are significant, especially when considering the large amount of SC typically processed in the BS.

Table 5.1: Computational Complexity in terms of the number of multiplications and additions for various detectors. The complexity of the F and G functions are shown in Table 5.2.

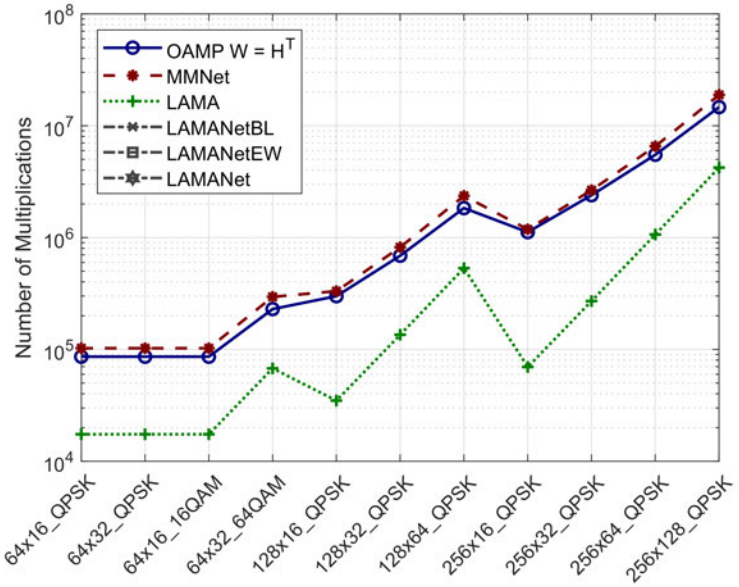
Detector	OAMP $\mathbf{W}_i = \mathbf{H}'$		MMNet $\mathbf{W}_i$ is learned		LAMA		LAMANetBL		LAMANetEW		LAMANet	
Algorithm	Algorithm 4				Algorithm 6				Algorithm 7			
Line	Mul	Add	Mul	Add	Mul	Add	Mul	Add	Mul	Add	Mul	Add
2					$M_R M_T + M_R M_T^2$	$M_R M_T + M_R M_T^2$						
4	$M_R M_T^2$	$M_R M_T^2 + M_T$	$2M_R M_T^2$	$2M_R M_T^2 + 2M_T$	M_T^2	$3M_T + M_T^2$	$2M_R M_T$	$2M_R M_T + M_R + 2M_T$				
5	M_T^3	$M_T^3 + 2M_T$	M_T^3	$M_T^3 + 2M_T$	$Mcompl F$	$AcompF$	$Mcompl F$	$AcompF$			$M_R M_T$	$M_R M_T + M_T$
6	$M_R^2 M_T$	$M_R^2 M_T + M_T$	$M_R^2 M_T$	$M_R^2 M_T + M_T$	$Mcompl G$	$AcompG$	$Mcompl G$	$AcompG$	$M_R M_T$	$M_R M_T + M_T$	$2M_R M_T$	$2M_R M_T + M_T$
7						M_T		M_T	$2M_R M_T$	$2M_R M_T + 2M_T$		
8	$M_R M_T$	$M_R M_T + M_R$	$M_R M_T$	$M_R M_T + M_R$	M_T	M_T	M_T	M_T	$Mcompl F$	$AcompF$	$Mcompl F$	$AcompF$
9	$M_R M_T$	$M_R M_T + M_T$	$M_R M_T$	$M_R M_T + M_T$					$Mcompl G$	$AcompG$		
10	$MCompF$	$ACompF$	$Mcompl F$	$AcompF$						M_R		
11	$M_R + 1$	$M_R + 1$	$M_R + 1$	$M_R + 1$					M_T	M_T		
12												
21	4	1	4	1								
per channel realization	$M_T^3 + M_R M_T^2 + M_R^2 M_T$	$M_T^3 + M_R M_T^2 + M_R^2 M_T + 4M_T$	$M_T^3 + 2M_R M_T^2 + M_R^2 M_T$	$M_T^3 + 2M_R M_T^2 + M_R^2 M_T + 5M_T$	$M_R M_T + M_R M_T^2$	$M_R M_T + M_R M_T^2$	-	-	-	-	-	-
per channel use	M_I (McompF+ $2M_R M_T$+ $M_R + 5$)	M_I (AcompF+ $2M_R M_T$+ $2M_R$+ $M_T + 2$)	M_I (McompF+ $2M_R M_T$+ $M_R + 5$)	M_I (AcompF+ $2M_R M_T$+ $2M_R + M_T$ + 2)	M_I (McompF+ McompG+ $M_T^2 + M_T$)	M_I (AcompF+ AcompG+ $M_T^2 + 5M_T$)	M_I (McompF+ McompG+ $2M_R M_T$+ M_T)	M_I (AcompF+ AcompG+ $2M_R M_T$+ $M_R + 4M_T$)	M_I ($Mcompl F$+ $Mcompl G$+ $3M_R M_T$+ M_T)	M_I (AcompF+ AcompG+ $3M_R M_T$ +M_R+ $4M_T$)	M_I (McompF+ $3M_R M_T$+ M_T)	M_I (AcompF+ $3M_R M_T$+ $2M_T$)

Table 5.2: Computational Complexity of F and G function

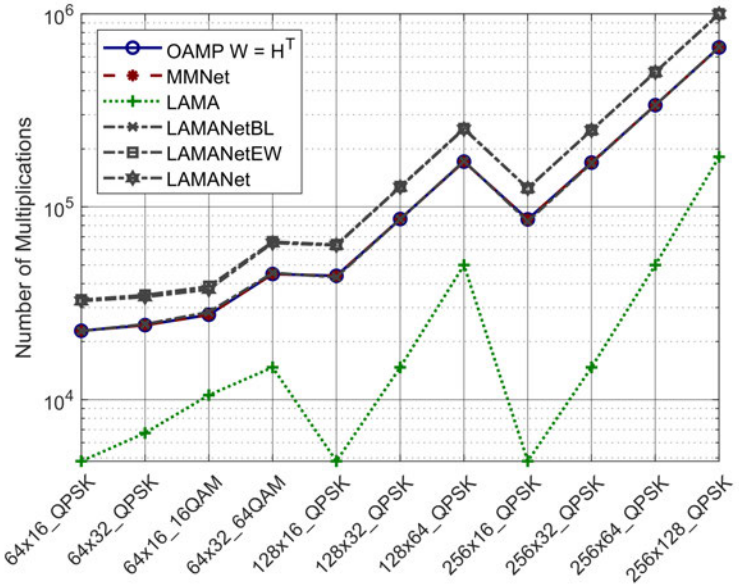
Line	F Function		G Function (in addition to F)	
	$MompF$	$AOmpF$	$MOmpG$	$AOmpG$
3	$2M_TM_C$	M_TM_C		
8		M_TM_C		
10	M_TM_C			
13	M_TM_C	M_TM_C	$M_TM_C + M_T$	$M_TM_C + M_T$
Total	$5M_TM_C$	$3M_TM_C$	$M_TM_C + M_T$	$M_TM_C + M_T$

Differences in the computations for each channel use between OAMPNet, MMNet, and LAMANet can be seen to be neglectable in Fig. 5.1b. However, only the OAMP type where $\boldsymbol{W}_i = \boldsymbol{H}^T; \quad i \in \{1, .., M_I\}$ is considered in this table. The other OAMP detector types will be more expensive, in particular since matrix inverses are required in each iteration. OAMP-Net only adds a few scalar parameters per interaction ontop of OAMP and therefore does not contribute significantly to the computational complexity. The introduction of element-wise extended detectors (LAMANetEW and LAMANet) shows an approximate increase in the number of multiplications by 32%. The simplifications introduced in LAMANet can reduce the computational complexity by up to 25% in some configurations such as 64x16_QAM1024. In other configurations with low modulation index, this difference is negligible; however, the hardware architecture can be simplified which is an advantage of LAMANet.

The LAMANetMMSE and LAMANetOpt detectors have the same computational complexity as LAMANet plus an additional term for their respective calculations. These calculations require a separate complexity analysis, as they highly depend on the chosen algorithm. For LAMANetMMSE this additional computation will be purely in the preprocessing, while for LAMANetOpt preprocessing and processing in the iterative part are affected. Hardware architectures for the LAMANetMMSE and LAMANetOpt algorithms are proposed in Section 5.3.2 and Section 5.3.3 respectively and evaluated in Section 5.5. For a complexity analysis of traditional massive MIMO detection algorithms, the interested reader is referred to [3].



(a) Number of multiplications in preprocessing performed once per channel coherence interval. LAMANet-type detectors have no preprocessing and therefore are not shown in the log-scale plot.



(b) Number of multiplications per channel use for the number of iterations $M_I = 10$.

Figure 5.1: The number of multiplications required for preprocessing and detection for various detector types, antenna configurations, and modulation types. The x-axis shows the real-valued system dimensions of $\boldsymbol{H}$ and the modulation type.

5.3 Accelerator Design Considerations

This section presents design considerations for implementing the LAMANet-MMSE and LAMANetOpt detector circuits. The targeted Xilinx FPGA and the Xilinx HLS design flow imply some of the design considerations. Xilinx HLS has been chosen as the implementation flow for faster development. However, it also has many limitations, and there remains the potential for optimisation by implementing LAMANet and its initialisation circuits in RTL. As dictated by the Tensorflow machine learning environment, the hardware design follows the transformation of the complex system to a real-valued system according to Section 4.2.2.

5.3.1 LAMANet

The LAMANet design shows high detection performance while reducing computational complexity in the preprocessing. The accelerator is designed according to the algorithmic description in Algorithm 7. The initialisation via MMSE-matrix in Algorithm 7 or the OAMP optimal matrix in Section 5.2.3 are just some possibilities. The LAMANet accelerator itself is agnostic to the type of initialisation matrix provided.

The accelerator is implemented as a non-iterative, deeply pipelined circuit for maximum throughput. As the LAMANet algorithm itself is iterative, the circuit may be deployed multiple times, each instance implementing one iteration. Alternatively, the output data might be fed back into the same instance of the accelerator to implement a low-resource design. The accelerator's high-level design is shown in Fig. 5.2.

The first step in any new iteration is to **store the input vectors** $\widehat{\boldsymbol{x}}$, and $\boldsymbol{y}$ and $\boldsymbol{H}_{stream}$ in internal memory. Ideally, this step would be omitted; however, the Vitis HLS design tool requires this step to correctly implement the pipeline via the "HLS DATAFLOW" pragma [141]. All data exchanged between pipeline stages must follow a single-producer, single-consumer approach, necessitat-

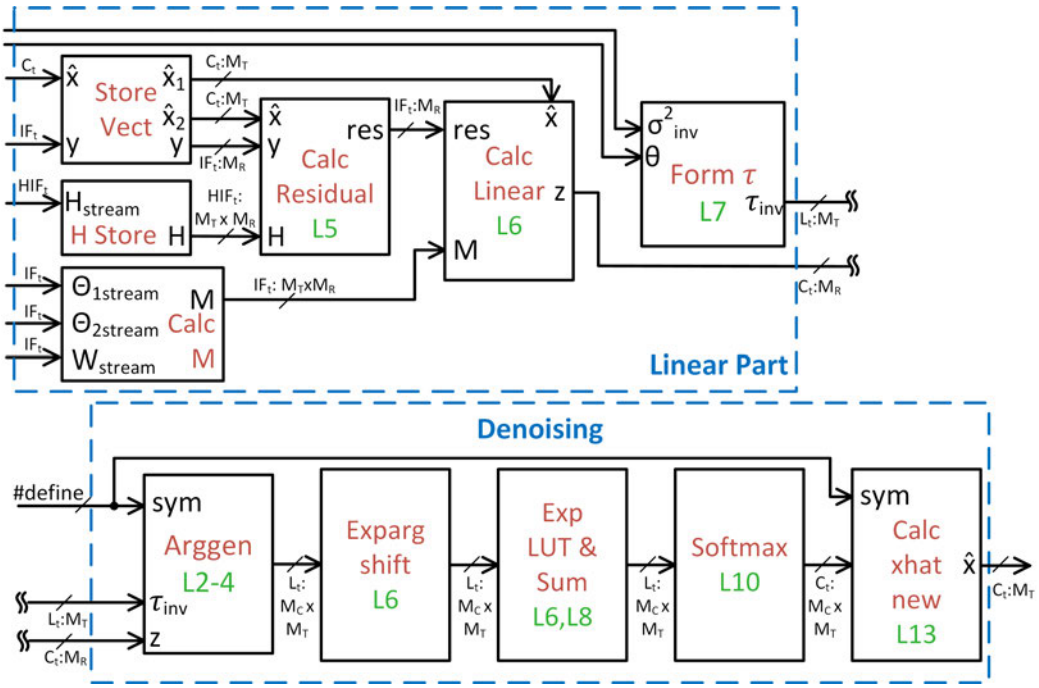


Figure 5.2: Pipeline structure of the LAMANet accelerator design. The green numbers within each pipeline stage refer to the line number in the respective algorithm marked by the blue boxes.

ing the duplication of $\widehat{x}$ in the "store-vect" block. Also, pipeline stages should not be bypassed for Vitis HLS to work correctly. This makes the "H store" block necessary to buffer $\boldsymbol{H}_{stream}$. The accelerator might be further optimised by implementing it in RTL.

As indicated in Fig. 5.2, we make use of four data formats throughout the design:

IF$_t$ smallest amount of bits, mostly used for inputs to the detector or to save memory space. In our design, this is an 18-bit signed fixed-point number of which 11 are fractional bits. The length of 18 bits is ideal for mapping to Xilinx memory elements and provides good accuracy.

IFH$_t$ same as $\mathrm{IF_t}$, but used to stream the H matrix. This format uses all 18 bits as fractional bits, as the absolute value of the entries of $\boldsymbol{H}$ are guaranteed by the optimal power control to be smaller than one.

C$_t$ typical format for computational results. In our design, this is a 27-bit signed fixed-point number of which 19 are fractional bits. This is ideal for Xilinx Ultrascale+ DSP slices as its multiplier can handle one input of up to 27 bits.

L$_t$ format for high-precision calculations with a large integer part. In our design, this is a 32-bit signed fixed-point number of which 12 bits are fractional.

Larger matrices in the design are streamed into the accelerator via the AXI-4 streaming protocol. This is to provide a standard interface without the need for extra buffering of data (such as in local memory). The sources of the streams might be other blocks in a receiver design or memory. We decide to implement a stream width of 32 values, i.e. 576 bits at 18 bits per value. For High Bandwidth Memory (HBM) accelerator cards, the bit-width of $\mathrm{IF_t}$ might be reduced to 16 bits to access an HBM memory port with 512 bits in parallel per stream (e.g. [142]).

In parallel with storing various input data, the streams $\boldsymbol{\Theta_{1stream}}, \boldsymbol{\Theta_{2stream}}$, and $\boldsymbol{W_{stream}}$ are provided to the accelerator. The stream's data is not stored but directly used to **calculate the intermediate variable** M. The number of parallel processing elements in this stage is a compile-time parameter and can be adjusted to match pipeline latency.

Next, the **residual is calculated** in one pipeline stage, and following that, the noisy tx-vector estimate $\boldsymbol{z}$ is calculated. It can be noted that in the LAMANet detectors, the Onsager term is not taken into account (Algorithm 7, Line 7). The **noise estimate** τ_{inv} in the next pipeline stage is formed simply by:

$$\boldsymbol{\tau_{inv}}[i] = \boldsymbol{\theta}[i] \times \sigma_{inv}^2, \quad i \in [0, M_T). \tag{5.17}$$

Next, the **denoising** as by Algorithm 3 is performed; however, only the denoised estimate $\widehat{\boldsymbol{x}}$ is produced. Calculating the noise variance ϕ is not required. The noise estimate σ^2 is used as an input to the accelerator to reduce the number of required divisions in the system to one per noise estimate. The Arggen pipeline stage also implements the required multiplication of τ_{inv} by two via an arithmetic left-shift (Algorithm 3, Line 3). It generates a 2D array by taking the difference between each entry of $\boldsymbol{z}$ and each possible symbol value. After squaring the result, τ_{inv} is multiplied by each corresponding row. Generating τ_{inv} instead of τ is not more complex but saves a significant amount of divisions in this stage.

The calculated 2D array is passed through the exponent function in the next step. Implementing this function in hardware would be too costly, so a lookup table approach is used instead. In the Expargshift stage, the values are prepared to be inputs to the lookup table as follows. First, the maximum of the current row (i.e. along the symbol axis) is found. The difference between the maximum input value of the lookup table and this maximum value is calculated. In the next step, this difference is added to each row value. This has the effect of shifting the values of the row into the input range of the lookup table. The maximum value of the row is placed at the maximum value of the lookup table. This way, the limited range of the lookup table is used most efficiently. It is clear that this linear shift in the input results in a non-linear shift in the exponent functions output; however since for this application, the difference between output values is important, this non-linear distortion is acceptable. In our experiments, we observe good matching with the ideal denoiser function. If an input value of the exponent function is below the minimum input value of the

lookup table despite the shifting, it is set to the lookup table's minimum value. The lookup table has 1024 samples and its input range is from -10.4 to 0. The minimum input corresponds to an output value of one LSB (2^{-15}). Each value of the LUT is quantised to 18 bits. This leads to the total memory usage of 1 BRAM18 (0.5 BRAM36).

In the next stage (ExpLUT & Sum), the output of the exponent lookup table is obtained and a row-wise sum is calculated. The reciprocal of each row-wise sum is calculated via a pipelined divider. This saves divisions in the next step. The Expargshift and ExpLUT pipeline stages are further pipelined with an interval of one clock cycle.

In the Softmax step, each value is multiplied by its respective, reciprocal, row-wise sum. This is the last step before obtaining the new $\widehat{\boldsymbol{x}}$ by multiplying the resulting 2D array with the symbol alphabet. The symbol alphabet is specified via compile-time defines. The newly calculated $\widehat{\boldsymbol{x}}$ is the final output of the accelerator and is specified in C_t-format for high accuracy. Evaluation results of the designed accelerator can be found in Section 5.5.

5.3.2 LAMANetMMSE

As mentioned above, the LAMANet detector supports many different forms of initialisation without changes to its structure. The MMSE initialisation provides a good compromise between computational complexity and performance. In order to evaluate the feasibility of the LAMANetMMSE detector, we present a possible implementation of a direct MMSE matrix calculation according to Algorithm 7, Line 2 and Line 3. It is worth noting that in regular massive MIMO detectors, there are more efficient ways to perform MIMO detection without directly calculating the inverse matrix [3]. However, direct inverse matrix calculation is one possibility. Neumann series expansion and Cholesky decomposition-based approaches have also been proposed [143, 144].

The proposed implementation is based on the matrix inverse in Xilinx's Vitis accelerated libraries, which uses the Cholesky decomposition [145]. The data type for all signals in the accelerator is single-precision floating-point [87]. Floating-point numbers are required for the matrix inverse to ensure algorithmic stability. The accelerator can use double-precision numbers by a compile-time switch, however, in our experiments, no significant performance improvements where observed as compared to single precision. The accelerator consists of three main stages as shown in Fig. 5.3.

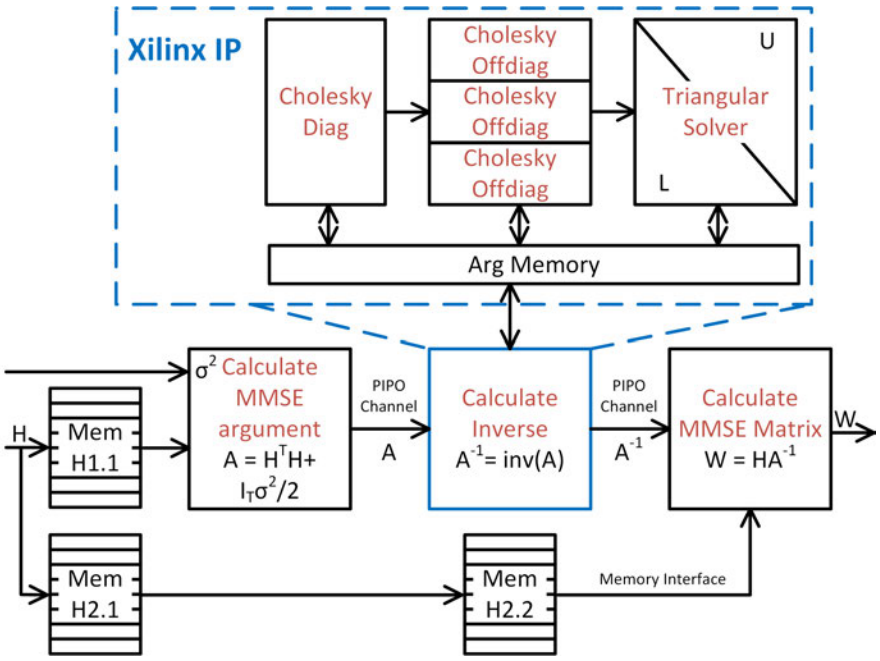


Figure 5.3: The architecture of Cholesky-based MMSE matrix calculation for initializing the LAMANetMMSE detector.

To satisfy the Vitis HLS tool's pipeline requirements, the H matrix is first stored in local memory, and in another temporary memory (MemH2.2). Then the input argument for the matrix inverse ($\boldsymbol{A}$) is calculated from the noise variance and the channel matrix. The resulting matrix is stored in a Parallel In Parallel Out (PIPO) channel for use in the inverse. Since $\boldsymbol{A}$ is positive definite, the Cholesky decomposition can be used to generate a triangular matrix such that $\boldsymbol{A} = \boldsymbol{LU}$, $\boldsymbol{U} = \boldsymbol{L}^T$;, were $\boldsymbol{A}, \boldsymbol{L}$, and $\boldsymbol{U}$ have real-valued entries. To generate the matrix $\boldsymbol{L}$, first, the diagonal entries are directly calculated in a pipelined fashion. Then, with a number of parallel off-diagonal calculation

units, all off-diagonal entries are calculated. The number of these processing cores is configurable during compile time. The $\boldsymbol{L}$ matrix is used to generate a column-wise intermediate signal $\boldsymbol{d}$ in a forward-substitution process. From this intermediate signal and $\boldsymbol{L}^T$ the inverse is computed in a backward substitution process:

$$\boldsymbol{L}_{\bullet,i}\boldsymbol{d} = \boldsymbol{I}_{\bullet,i};\ \boldsymbol{L}^T_{\bullet,i}\boldsymbol{A}^{-1}_{\bullet,i} = \boldsymbol{d};\ i \in [0, M_T);\ \boldsymbol{d} \in \mathbb{R}^{M_T}. \tag{5.18}$$

The process is repeated for each column of the inverse matrix $\boldsymbol{A}^{-1}$. Finally, the MMSE matrix is calculated by $\boldsymbol{W} = \boldsymbol{H}\boldsymbol{A}^{-1}$.

5.3.3 LAMANetOpt

For the option of initializing LAMANet with the optimal matrix, the eigenvalue decomposition of the Gram matrix is performed once per channel realisation according to Algorithm 8. The proposed design first calculates the Gram matrix ($\boldsymbol{G} = \boldsymbol{H}^T\boldsymbol{H}$) and then the eigenvalues and eigenvectors based on an IP from Xilinx's Vitis accelerated libraries [145]. The eigenvalue decomposition uses the one-sided Jacobi decomposition as the Gram matrix is symmetric. Off-diagonal entries of the matrix are eliminated in an iterative process using 2x2 Jacobi-Rotations [146]. A schematic representation of the circuit is shown in Fig. 5.4.

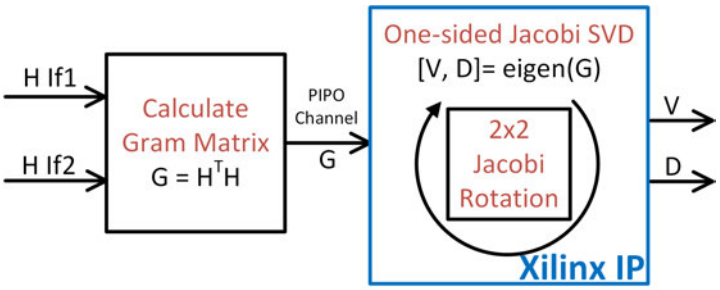


Figure 5.4: The architecture of the Jacobi-based eigenvalue decomposition calculation for initializing the LAMANetOpt detector.

The LAMANetOpt detector itself has to be modified according to Algorithm 8 to make use of the obtained eigenvalues and eigenvectors. For this purpose, a streaming interface for the eigenvectors $\boldsymbol{V}$ consisting of 32 $\mathrm{IF_t}$ entries is implemented. Another stream with $\boldsymbol{H}$ is provided to calculate $\hat{\boldsymbol{W}}_i$. The eigenvalues are provided via a standard interface. The calculations are implemented in a pipelined fashion, first the reciprocal of $\boldsymbol{M}_i$ is calculated in two stages, then it is multiplied with the eigenvalues in another three stages to calculate $\hat{\boldsymbol{W}}_i$ according to Eq. (5.15). However, v_i^2 is not multiplied for reasons of numerical stability in the case of small values of v_i^2. This does not influence the final result $\boldsymbol{W}_i$ as the division of $\hat{\boldsymbol{W}}_i$ with $\mathrm{tr}(\hat{\boldsymbol{W}}_i\boldsymbol{H})$ cancels the multiplication with v_i^2 regardless. Finally, $\hat{\boldsymbol{W}}_i$ is calculated and provided for further processing to the unchanged rest of the algorithm. The performance results are presented in Section 5.5.

5.4 Detection Performance

In this section, we present and compare the detection performance of the proposed neural networks. The results are obtained via Tensorflow simulations. The Tensorflow framework is based on [147], although significant alterations have been made to the framework, and the new LAMANet detectors have been implemented. Python 3.6.9 and TensorFlow 1.13 are used. Similar to the work in [26], a realistic channel model is generated via QuaDriGa with the parameters as shown in Table 5.3. Without loss of generality, we choose a configuration of 64 base station antennas and 16 mobile users for detection performance measurement ($M_R = 128$, $M_T = 32$). For the hardware implementation in Section 5.5, more configurations are evaluated. For each detector, the results for QPSK, QAM16 QAM64, QAM256, and QAM1024 are provided. The behaviour of the detectors is similar regardless of the modulation scheme.

Table 5.3: Parameters for detection performance simulation and hardware test data generation.

Property	Value
Channel Bandwidth	20 MHz
Center Frequency	2.53 GHz
Subcarriers	1024 subsampled to 256
MR	128
MT	32
QuaDriGa Scenario	3GPP_3D_UMa_NLOS
User Distance Min	10
User Distance Max	500
Sector Angle	120°
User speed	1 m/s
Coherence time	59.3 ms
Number of steps along trajectory	20
Drops (number of random user placements)	2
Total number of channel realisations	10240

Untrained channel realisation

First, the performance of the untrained detectors is compared. It is important that the detector can perform reasonably well in an untrained state such that the training process on a specific channel can be removed from the critical path (i.e. does not contribute to the latency) of the detector. This is the main advantage of the LAMANetMMSE and LAMANetOpt detectors over the MMNet detector. Out of the 5120 channel realisations, 2000 random channel realisations are selected per test and, per tested SNR. The test is repeated 10 times for each SNR value to gain reliable SER numbers also in the low SER case. All the SERs values are averaged per SNR value and shown in Fig. 5.5.

MMNet and LAMANetBL perform poorly in the untrained state giving SERs in the range of 0.5-0.9. This loss in performance for MMNet is easily understandable as the matrix in the linear part has to exactly match the current channel realisation to perform the noise shaping correctly. Also, the classical IO-LAMA [117] algorithm performs poorly. On the other hand, the approaches which initialise $\boldsymbol{W}_i$ with a sensible guess (such as the MMSE matrix or the OAMP optimal matrix) maintain much better SER performance. The learnable parameters $\boldsymbol{\Theta}_{i,1}, \boldsymbol{\Theta}_{i,2},$ and $\boldsymbol{\theta}$ can be initialised to the identity element of their respective operators in order to avoid any influence in the untrained state. This is why LAMANetMMSE and LAMANetOpt achieve approximately the same per-

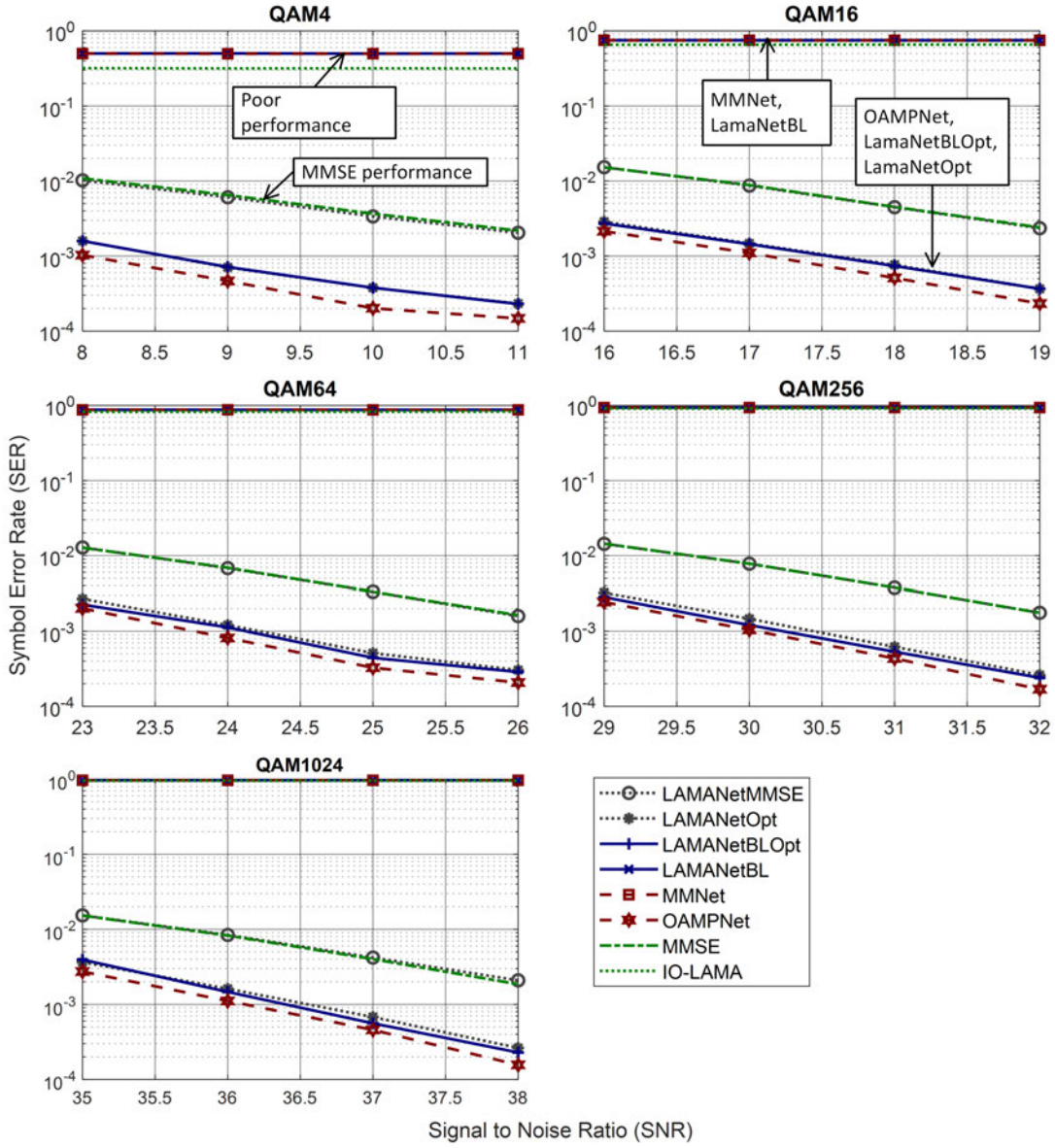


Figure 5.5: Untrained detector performance on 20000 randomly taken channels from one drop (5120 channel realisations). The configuration of $M_R = 128$, $M_T = 32$ for various modulations is shown.

formance as MMSE and OAMP detectors even in the untrained state. Fig. 5.5 shows that even when untrained, the performance of the simplified LAMANet detectors (LAMANetMMSE and LAMANetOpt) is not degraded as compared to the LAMANet baseline detector (LAMANetBLMMSE and LAMANetBLOpt). This further confirms the feasibility of the LAMANet initialisation approach.

Trained channel realisation

Each detector is trained for 900 epochs with a batch size of 300 samples at the highest SNR value of the respective range. The loss is formed by comparing the actually transmitted symbol with the prediction of the detector. Based on the loss, the learnable parameters are updated in the back-propagation step of the Adam optimiser. For testing, the batch size is increased to 10,000 samples (limited by GPU memory) and the results are averaged over 10 epochs. The measurement is repeated for each SNR value separately. All the detectors are trained and tested on the same, randomly chosen channel realisations for a configuration of $M_R = 128$, $M_T = 32$, and various modulation types in Fig. 5.6. It can be seen that OAMPNet cannot quite reach the performance of LAMANet and MMNet detectors. Whether an MMSE matrix or the OAMP optimal matrix is used for $\boldsymbol{W}_i$ has no significant impact on the fully trained LAMANet detector. At higher SNR values LAMANet and LAMANetBL show a small performance loss as compared to MMNet.

To investigate this difference further a Cumulative Distribution Function (CDF) is shown in Fig. 5.7. The trained detection performance is obtained over 143 distinct channel realisations. The detectors are trained for each channel realisation for 9,000 epochs and the performance is evaluated across 9,000 channel uses. A small performance loss can be observed between MMNet and LAMANetBL, which confirms that an AMP-based design suffices to achieve high performance if the algorithm is enhanced with machine learning elements. The detection performance is almost the same for LAMANetBL and LAMANet. This confirms that the learned parameters can compensate for the missing Onsager and tau-scaling calculations.

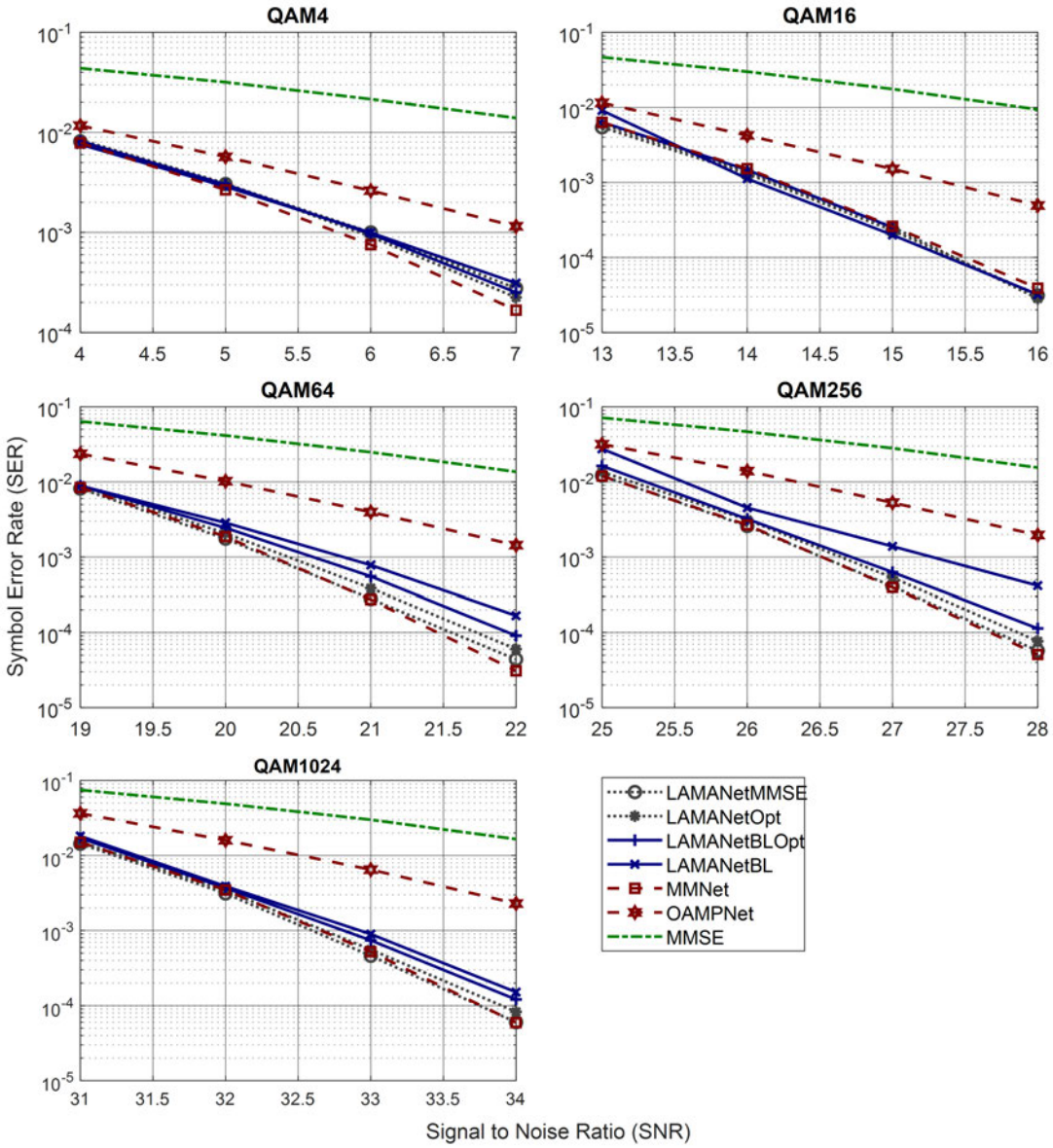


Figure 5.6: Trained detector performances on a randomly selected channel for the configuration $M_R = 128$, $M_T = 32$.

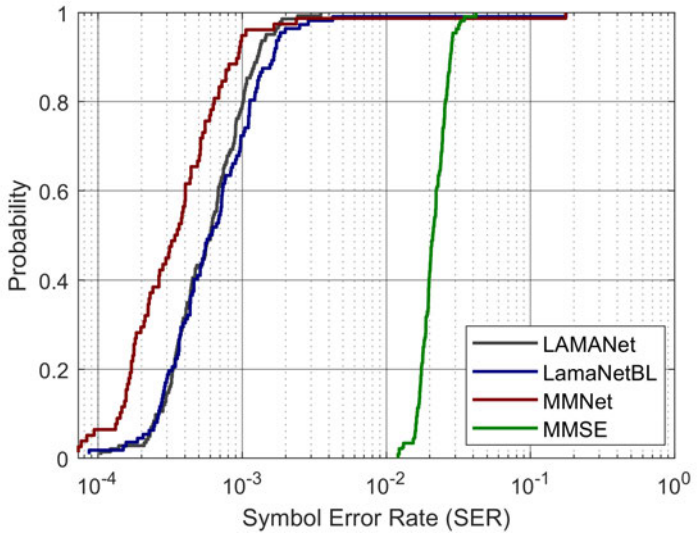


Figure 5.7: CDF of the trained detector performance over a total of 143 channel realisations for a configuration of $M_R = 64$, $M_T = 16$, QAM4. The measured SNR is ≈ 7 dB.

5.5 Circuit Performance

The baseband processing for massive MIMO in a 5G-NR-type radio access network can be located either directly in the Distributed Unit (DU) - close to the cell antennas or in a Centralised Unit (CU) [148, Section 11]. This choice influences the type of accelerator deployed for processing. In the following evaluation, we choose Alpha Data's RF-SoC [7], which is most suitable for deployment in DUs due to its integrated RF chains. More information on the FPGA board is given in Section 2.5.1.

We first analyse and present latency, throughput, and utilisation results for the LAMANet detector and its preprocessing circuits. Latency and throughput are defined in Section 2.5.1 and are obtained by Register Transfer Level (RTL) simulation. Utilisation and timing results are obtained after out-of-context P&R in Xilinx Vivado. The maximum clock frequency is estimated after P&R of the circuit. The target frequency set during synthesis is 300 MHz. This is an important parameter to set as it influences the level of optimisation during synthesis and P&R. For Ultrascale+ Architectures 300 MHz is a moderate frequency and should be reachable without trouble. If synthesis cannot reach this

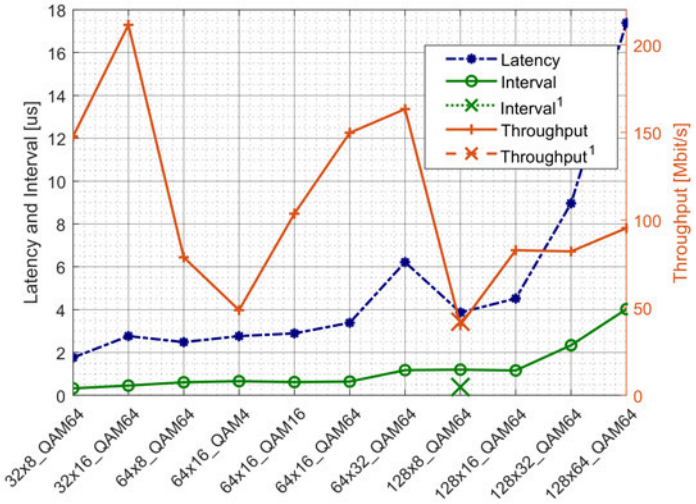
frequency, it might indicate design issues such as overly long critical paths or bad placement. The actually achieved clock frequency after P&R is used for throughput calculations. The presented configurations show the number of antennas, i.e. the complex system dimensions. After real decomposition, M_R and M_T are twice these values.

It is important to functionally verify the proposed accelerators. For the LAMA-Net accelerator, the Tensorflow implementation is used as a golden model. During the running of the Tensorflow evaluation, accelerator input and output data for each configuration and for each tested SNR value after one iteration and 10 iterations are exported. For all tests, we follow the design strategy in Vitis HLS. First, the functionality is verified in C-simulation, then, after high-level synthesis, the correctness of the RTL is verified in digital circuit simulation. The MMSE preprocessor is evaluated by comparing the MMSE matrix calculated in Tensorflow with the hardware calculated MMSE matrix. The eigenvalue decomposition preprocessor is verified by recreating the Gram matrix from the accelerator's outputs. Then the Gram matrix is calculated in the testbench and compared with the accelerator's output.

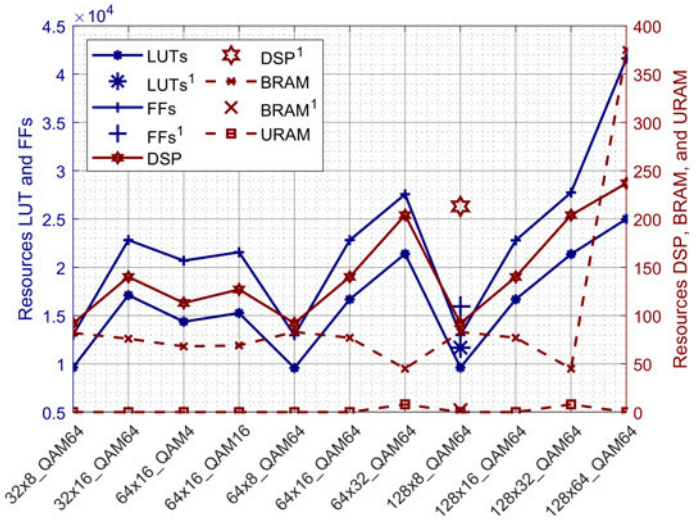
LAMANet

The LAMANet detector as described in Section 5.3 is profiled for various configurations. The results are shown in Fig. 5.8, where the complexity of the configurations is increasing from left to right. The latency and interval are fairly constant for all configurations. The throughput is in the range between ≈ 40 Mbit/s and ≈ 210 Mbit/s, peaking at the largest ratios of $\overline{M_R}$ to $\overline{M_T}$, except for the configurations to the right, where the increase in interval time dampens the achievable throughput.

A classical AMP accelerator for FPGA deployment is proposed in [4]. Its interval and throughput are plotted in Fig. 5.8a for comparison. Since it implements three iterations, the reported throughput is multiplied by three and the reported interval is divided by three to ensure a fair comparison with one iteration of



(a) Latency and throughput



(b) Resource utilisation

Figure 5.8: LAMANet performance metrics for various antenna and modulation configurations. The numbers on the x-axis indicate the number of antennas. M_R and M_T are twice the indicated values. [1] Results from [4]. Interval and throughput are normalised to one iteration. [4] only supports QPSK, not QAM64.

Table 5.4: Detailed HLS results for the pipelined LAMANet 64x16_QAM64 design.

Modules	Latency [clk]	Interval [clk]	BRAM	DSP	FF	LUT
|=|=|=| LAMANet Top |=|=|=|	531	143	50 (2%)	140 (3%)	23739 (2%)	12629 (2%)
Store xhat	34	34	-	-	27	87
Store y	130	130	-	-	601	218
Store H	130	130	-	-	603	172
Calculate M	132	132	-	32	1239	131
Form τ	37	37	-	2	347	175
Calculate Residual	142	143	-	32	3528	464
Store residual	66	66	-	-	583	708
Calculate z = M x residual	133	133	-	32	1813	1293
Calculate z = z + xhat	36	36	-	-	1866	2334
Calculate Arggen	71	71	-	16	2048	914
Calculate Exparg shift	41	41	-	-	1027	1095
Calculate Exp, LUT, & Sum	103	74	3	16	4739	3788
Calculate Softmax	134	134	-	2	168	250
Calculate xhat new	140	139	-	8	262	403
Assign xhatnew to output	35	35	-	-	11	87

LAMANet. The work in [4] has a slight performance advantage in terms of interval time and throughput. It is worth noting that only QPSK is supported in the reference detector. Also, it uses the compacted linear form (with $\boldsymbol{G}$ and $\boldsymbol{y_{MRC}}$) and does not support ML-enhancements.

The number of connected user antennas largely determines the number of resources used by LAMANet. The choice of using BRAM or URAM resources is left to the high-level synthesis tool. Except for the last configuration, the BRAM usage is fairly constant at ≈ 35 BRAM36 blocks. The used LUT, FF, and DSP resources are largely determined by the number of connected users. Compared to the work in [4], less than half the DSP resources are used. LUT, FF usage is comparable in both designs. LAMANet needs more memory to store streamed matrices in the accelerator, this is due to the HLS design limitations as discussed above.

The detailed results after high-level synthesis for the configuration of $\overline{M_R} = 64$, $\overline{M_T} = 16$, and QAM64 modulation is shown in Table 5.4. The Modules largely correspond to the pipeline structure in Fig. 5.2. It can be seen how the interval of many pipelined stages is close to the maximum of 143 clock cycles, which is ideal.

Table 5.5: Accelerator performance comparison for 128x8 antennas (complex system dimensions)

Detector	LAMANet		AMP [4]	NSE [149]	PGS [150]
FPGA Device	Ultrascale+ xczu27dr-2	Zynq7 xc7z045-2	Virtex-7 xc7vx690t		
M_I	1	3	3	3	3
Modulation	QAM64	QAM64	QPSK	QAM64	QAM64
LUTs	9609	17478	11673	148797	10085
FFs	12990	25699	15943	161934	12074
DSP48s	92	165	213	1016	266
BRAMs	83	75	2	16	4
Interval [clks]	399	1203	496	196	204
Clock Frequency [MHz]	326	288	429	317	322
Throughput [Mb/s]	39	12	14	621	76
Throughput/LUTs	5442	877	1186	4173	7536
Real-time SC	762	670	807	1509	1473

When comparing the proposed LAMANet accelerator with previous work in Table 5.5, the performance is competitive even though the proposed accelerator implements machine learning enhancements. The design is synthesised for a Xilinx Zynq7 device for a fair comparison with the other work. When compared to the work in [4], which implements a similar AMP algorithm, our design achieves comparable throughput and throughput per used LUTs. It uses comparatively more BRAM resources but fewer DSP resources and supports QAM64 modulation. Other detector types, such as the Neumann Series Expansion (NSE) [149] and Parallel Gauss-Seidel (PGS) [150] detectors tend to outperform the AMP-type detectors. However, they just implement linear detection whose SER performance is lower. We noticed some inefficiencies created by the HLS tool when targeting the Zynq7 platform, such as more LUTs and more DSPs used. The implementation results on current FPGA hardware (Ultrascale+ RFSoC) are provided for reference.

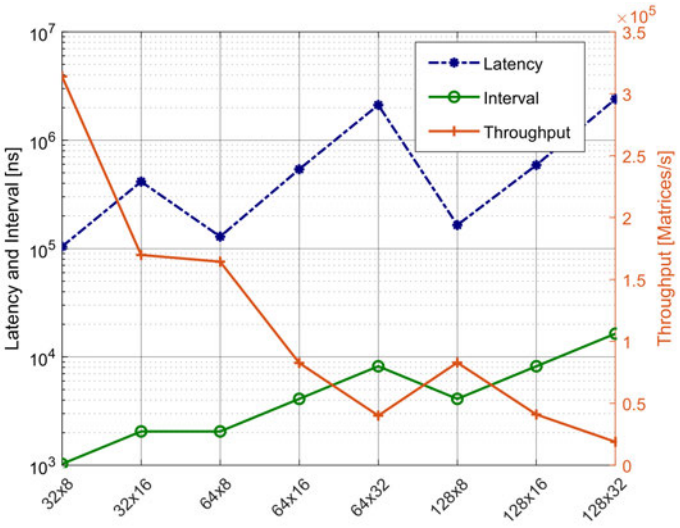
The 3GPP technical specification TR 38.913 defines the target for user plane latency depending on the use case [151]. For enhanced Mobile BroadBand (eMBB), this latency is defined as 4 ms in the downlink and 4 ms in the uplink. However, the processing delay for the base station is just one part contributing to the overall delay and the standards do not provide delay requirements for sub-components such as detector circuits. In the 5G-NR numerology, one slot consists of 14 symbols in the case of regular cyclic prefix length [96]. The

target delay for the base station processing is one slot according to [57]. The duration of one slot depends on the subcarrier spacing and ranges from 1 ms for a subcarrier spacing of 15 kHz to 62.5 µs for a subcarrier spacing of 240 kHz. The last row of Table 5.5 shows the number of subcarriers that can be processed in real-time in one slot of 15 kHz spaced subcarriers.

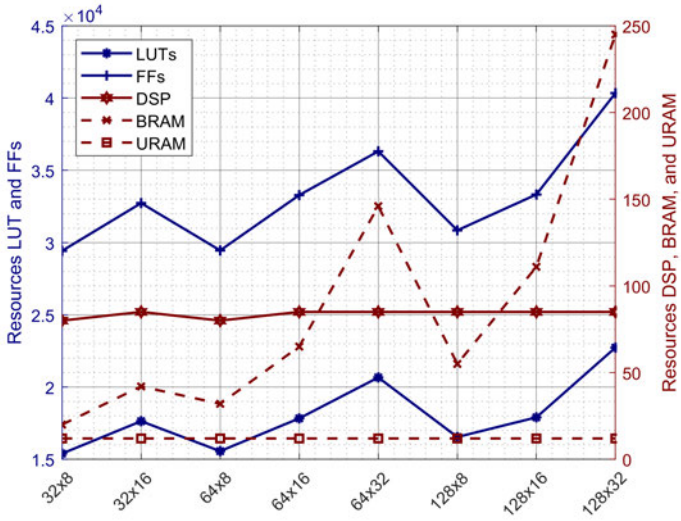
LAMANetMMSE

The right initialisation of the LAMANet detector is important for its untrained detection performance. The option of initialising LAMANet with an MMSE matrix provides a good trade-off between computational complexity and untrained detection performance. In Fig. 5.9a and Fig. 5.9b the latency and resource numbers are shown respectively for the MMSE matrix generation circuit presented in Section 5.3.2. The throughput decreases from $\approx$314 k Matrices per second in the smallest system configuration (32×8) to $\approx$19 k Matrices per second in the largest configuration (128×32). The resource usage in terms of DSPs and URAM storage is almost constant for all configurations, while BRAM usage increases strongly in the larger configurations. LUT and FF utilisation do not vary significantly considering the large number of these resources available in modern FPGA devices.

The walking speed of 1 m/s in the SER simulation setting (Table 5.3) results in a coherence time of ≈ 9.3 ms when approximating it via $T_c = \frac{\lambda}{2v}$, where λ is chosen as the center frequency for this approximation and v is the UT's speed [2]. Using this relationship, Table 5.6 shows the number of subcarriers one instance of the MMSE circuit is capable of processing. For MMSE preprocessing, the number of matrices calculated per coherence interval is sufficiently large to process several thousand subcarriers with one preprocessing circuit even for larger antenna configurations. This tolerance can allow for higher user speeds too.



(a) Latency and throughput



(b) Resource utilisation

Figure 5.9: MMSE matrix calculation performance for various antenna configurations according to the design in Section 5.3.2. The numbers on the x-axis indicate the number of antennas. M_R and M_T are twice the indicated values.

Table 5.6: Number of preprocessing matrices per coherence time according to the settings in Table 5.3. The antenna configuration $2M_R \times 2M_T$.

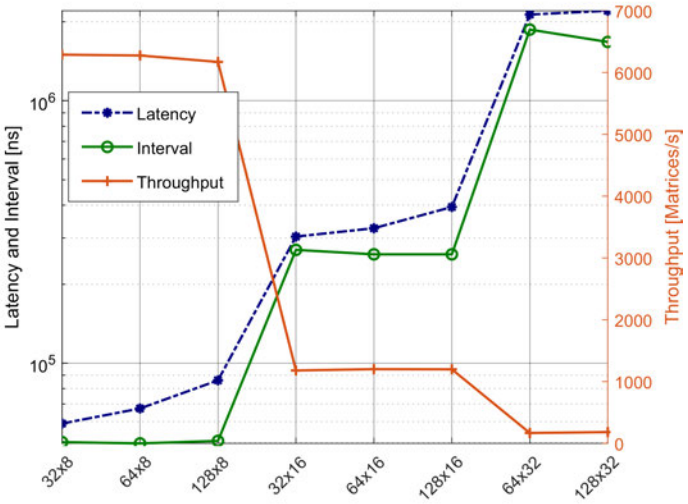
	MMSE Matrix		Eigendecomposition	
Antenna config.	matrices/ sec	matrices/ coherence time	matrices/ sec	matrices/ coherence time
32x8	314k	18616	6289	372.9
32x16	170k	10069	6276	372.1
64x8	164k	9746	6172	366.0
64x16	82k	4897	1180	70.0
64x32	40k	2376	1201	71.2
128x8	82k	4915	1198	71.1
128x16	41k	2431	168	10.0
128x32	18k	1120	183	10.8

LAMANetOpt

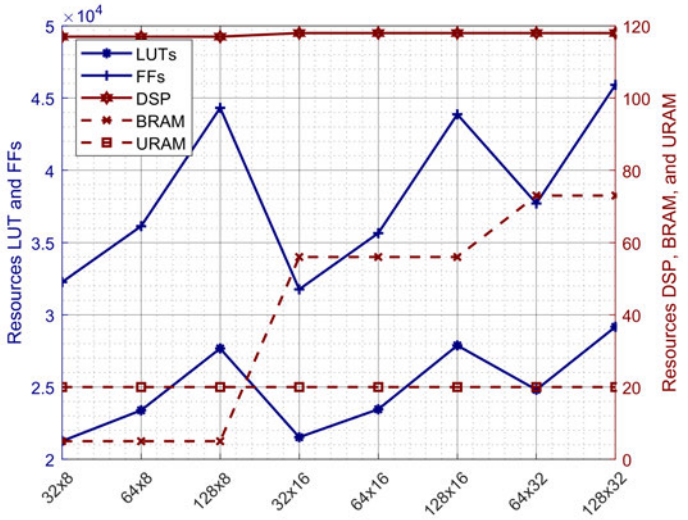
To achieve better SER performance in the untrained detector, it is possible to calculate the optimal initialisation matrix according to Section 5.3.3. The throughput and resource usage for the required eigenvalue decomposition are shown in Fig. 5.10a and Fig. 5.10b respectively. The resource usage, as well as the throughput, are highly dependent on the number of users connected. This is understandable as the majority of the computational work is spent on the eigenvalue decomposition of the Gram matrix $\boldsymbol{G}$ which is a $M_T \times M_T$ square matrix.

For the eigenvalue decomposition accelerator, the number of channel matrices per coherence interval is substantially lower than in the MMSE case. Considering a large number of SC such as 1200 in the $15\,\mathrm{kHz}$ SCS case, and at the relatively low speed of $1\,\mathrm{m/s}$, only the smaller antenna configurations can be processed without instantiating multiple preprocessing modules. As the choice of the initialisation matrix influences the detector performance only in the untrained state, the hardware cost of optimal processing might be too high in some cases.

The extension of the detector itself is costly as Table 5.7 shows. For an example antenna configuration of 64x16 (real-valued matrix dimensions are 128x32), the required DSP resources more than double and the BRAM resources increase more than tenfold. As we limit our DSP usage per pipeline stage to 32



(a) Latency and throughput



(b) Resource utilisation

Figure 5.10: Eigenvalue decomposition performance metrics for various antenna configurations according to the design in Section 5.3.3.The numbers on the x-axis indicate the number of antennas. M_R and M_T are twice the indicated values.

Table 5.7: Comparison of LAMANet and LAMANetOpt accelerator for 64x16 antennas (Complex system dimensions)

Detector	LAMANet	LAMANetOpt
FPGA	Ultrascale+ RFSoC xczu27dr-2	
M_I	1	1
Modulation	QAM4	QAM4
LUTs	14363	24296
FFs	20681	33677
DSP48s	113	255
BRAMs	34	364
Latency [clks]	202	6225
Clock Freq. [MHz]	307	303
Throughput [Mb/s]	48.6	1.6
Throughput/LUTs	3383	66

DSPs, the computation of $\hat{\boldsymbol{W}}_i$ from $\boldsymbol{H}$, $\boldsymbol{V}$, and $\boldsymbol{M}_i$ in Eq. (5.15) is identified as the bottleneck taking $M_T^2 M_R/32 = 4096$ clock cycles. Since the interval will be limited by this, other processes in the pipeline are allowed to also take this maximum time to reduce resource usage. The throughput of the accelerator is lowered to ≈ 1.6 Mb/s as compared to the ≈ 48.6 Mb/s in LAMANet.

5.6 Summary

Based on the previous work in the literature, this chapter introduced a novel model-driven machine learning algorithm named LAMANet. LamaNet is computationally simpler as previously proposed algorithms such as TISTA, OAMP-Net, and MMNet as it is based on the simpler AMP detector instead of the OAMP detector. Further algorithmic simplifications in LAMANet reduce the computational complexity and allow for efficient implementation in hardware. Nevertheless, we found that these simplifications do not degrade the SER performance of LAMANet in the trained state of the detector due to the learnable parameters. LAMANet shows comparable SER performance to MMNet.

Some of the previous proposals such as MMNet perform very poorly in the untrained state which makes practical deployment difficult. This chapter introduced a novel way of incorporating learnable parameters into the algorithm, allowing the detector to be initialised accordingly and to perform at a baseline level even when untrained.

In the integrated receiver, some training of LAMANet will still be required while being deployed. This training effort is not part of the main processing chain of the receiver and can be removed from this critical path of the system. Nevertheless, an efficient way of providing new weight matrices to the accelerator needs to be considered during integration. The online training architecture needs to be investigated in future steps.

Based on the algorithmic exploration, hardware architectures for Xilinx Ultrascale+ FPGAs are proposed and implemented. The LAMANet accelerator is comparable to previous work in the literature in terms of FPGA resource utilisation, latency, and throughput while supporting machine learning enhancements to increase detection performance.

Further, the preprocessing circuits to initialise LAMANet are implemented on FPGA hardware and profiled. In particular, LAMANetMMSE showed the feasibility to process a large amount of SC in real-time. LAMANetOpt uses the optimal initialisation matrix, however, it is computationally more challenging. The work presented in this chapter is important as it takes a step towards deployable machine learning-enhanced massive MIMO detection in real-time.

Chapter 6

Conclusions and Future Research

In this thesis, we took steps towards real-time, ML enhanced signal detection in mobile communication systems such as 5G-NR. In particular, the core signal detection tasks of OFDM and massive MIMO were investigated. As shown in the literature, the presented research confirmed that the ML method can offer higher detection performance compared to traditional detection algorithms. However, the core contribution of this work is in taking steps towards real-time and at-scale processing of some of the most promising ML detectors. This is achieved by a few means a.) the detailed analysis of current ML detectors in terms of algorithmic complexity and the possibilities of algorithmic advancements or simplifications, b.) optimisation of the algorithm for hardware deployment, c.) development of high-performance FPGA architectures, and d.) implementation and verification via digital circuit design. The following sections provide conclusions and future research directions for the OFDM signal detector of Chapter 3 and the massive MIMO signal detectors of Chapter 4 and Chapter 5.

6.1 OFDM Signal Detection

Chapter 3 introduced a fully connected neural network based OFDM signal detector. The design was inspired by [16]; however, algorithmic improvements significantly reduced the computational complexity of the network by $\approx 4.02\times$. Further, the technique of deep compression was utilised to achieve an overall reduction in memory size of $\approx 27\times$. Since the fully connected network is

memory bound, this is particularly important. With a BER of 7×10^{-3} at 25dB SNR and the same neural network structure, we could not reproduce the results of [16] in our environment. However, from this baseline, the new algorithm and the deep compression approach (quantisation and Huffman coding only) introduce a performance loss of $\approx 2.5dB$ at a BER of 1.4×10^{-2}.

Based on the new algorithm, a hardware accelerator specifically designed for the detector workload was proposed. Throughput calculations of 13 accelerators based on the synthesis and simulation results of the on-chip and off-chip accelerators showed that up to 832 SC can be processed in real-time on the targeted Xilinx RFSoC FPGA platform. The detector only supports QPSK modulation which leads to a total throughput of $\approx 25Mbits/s$. The average latency is ≈ 3.67 symbols ($\approx 244\mu s$) while the maximum latency is ≈ 11.52 symbols ($\approx 768\mu s$).

When comparing the resource utilisation with traditional approaches, a gap in the used DSP and memory resources can be observed. While some classical least square channel estimation and zero-forcing need only 30 DSPs and 1.5 BRAMs, one URAM neural network accelerator requires 80 DSPs, 49 BRAMs, and 12 URAMs blocks. It showed the high cost of the neural network based detection.

The implemented detector is one of the first real-time capable detectors proposed in the literature. This makes it valuable as it shows the feasibility of real-time neural network detector processing. At the same time, many limitations of the current design can be pointed out as presented in Section 3.7. On the algorithm level, we see the following main limitations.

The OFDM pilot symbols are not assigned according to any standard, as the neural network requires comb-type pilot assignment. More advanced architectures are capable of time series processing, such as LSTM networks or model-driven approaches, and might help (e.g. ComNet [24]). Also, higher modulation orders might be easier supported in model-driven methods. Our neural network detector currently only supports QPSK modulation. By performing bitwise sig-

nal detection, higher modulation orders are possible; however, this increases the number of output neurons significantly. The alternative of having multi-level activation functions in one or more of the final layers (such as reported in [105]) proved to be unsuccessful in our experiments.

Although the real-time capability of the FC neural network detector has been shown, doubts on the feasibility of data-driven designs remain. Fundamentally, removing all domain knowledge from the detection process has been shown to lead to an overly high computational complexity of the detector. Even though data-driven designs are appealing conceptually, large-scale, real-time deployment is still not achievable with current processing capabilities. An increase in processing power over time might not even help the situation, as the requirements for mobile communication systems and the requirements for the detection process might also increase over time. Model-driven designs such as those used for the massive MIMO detection task are a more suitable alternative.

6.2 Massive MIMO Signal Detection

Massive MIMO is investigated in Chapter 4 and Chapter 5. The former reviews a wide range of ML detectors and implements the first version of a hardware accelerator for algorithms of the OAMP family. A review of previously proposed data-driven detectors showed that for real-time processing, these approaches are not suitable. The computational complexity is as high as $\approx 312k \times M_R \times M_T$ multiplications per symbol in some proposals. Even if real-time processing would be possible, the achieved gain in detection performance might not justify such a complex approach. For this reason, Chapter 4 and Chapter 5 focus on model-driven detectors.

OAMP detectors are one of the most investigated detectors and are comparatively simple in terms of concept and computational complexity. Chapter 4 introduces an OAMP accelerator and profiles it for system dimensions ranging from 32 to 128 base-station antennas. In the smallest system dimensions

of 32 base station antennas in combination with eight connected users, the accelerator achieves a pipelined latency of $1.74\mu s$ while using 106 DSP and 10.5 BRAM primitives per iteration. When compared to a classical AMP-type detector the performance of the accelerator is lower. For a configuration of 128 base station antennas with eight connected users, the throughput is around 5 Mbps, while a classical detector achieves around 14 Mbps while performing three iterations of the algorithm. This is due to some inefficiencies in the OAMP algorithm compared to the classical algorithm and due to the fact that no machine learning enhancements are supported.

Chapter 4 showed the shortcomings of current ML approaches and led to LAMANet, which is presented in Chapter 5. LAMANet solves two problems which made previous ML-based OAMP detectors infeasible for practical deployment.

First, the computational complexity is reduced significantly compared to OAMP-Net as LAMANet does not require a matrix inversion per iteration. Also compared to MMNet, LAMANet eliminated the large amount of preprocessing required by using the simpler AMP algorithm. Training on 143 different channel realisations shows a loss in average SER between MMNet and LAMANet of just $\approx 2.6 \times 10^{-4}$ at a training SNR of 7dB and a configuration of 32 base station antennas with eight connected users. This validates the LAMANet approach and confirms that using the complex OAMP algorithm is not required if the algorithm is enhanced with ML components.

Second, LAMANet mitigates the online training requirement found in MMNet. By initialising the detector with either the MMSE matrix or the OAMP optimal matrix, the respective MMSE or OAMP optimal performance can be achieved in the untrained state. This does not have a significant adverse influence on the trained performance. LAMANetMMSE and LAMANetOpt thereby reach MMNet-level performance.

The proposed LAMANet accelerator can achieve a throughput of up to ≈ 210 Mbit/s in a configuration of 32 base station antennas and 16 connected users with a modulation scheme of QAM64. In a setup of 126x8 antennas, LAMANet achieves a throughput of 39 Mbit/s while the classical AMP detector archives 42 Mbit/s per iteration. LAMANet however is profiled on a newer Ultrascale+ FPGA. In terms of resource usage, the LAMANet accelerator uses less than half of the DSP resources than the classical design.

The LAMANetMMSE initialisation circuit reaches a throughput of up to $\approx 300k$ channel matrices for the antenna configuration of 32x8. This makes it suitable for deployment in the receiver. The LAMANetOpt initialisation circuit on the other hand only reaches $\approx 6k$ channel matrices for this antenna configuration.

The biggest challenge for LAMANet-type detectors is the still required online training. Hyperlearning-based detectors might offer a solution; however, in our experiments, with our channel models, we could not reproduce the generalisation property reported in [124]. Further investigation into this is required. The challenge with other detectors is still the computational complexity - particularly in data-driven designs.

6.3 Future Research Towards Realising a Demonstrator

This thesis covered various topics, from algorithmic link-level simulations to digital hardware design. This range of topics was necessary to realise the goal of taking a step towards deployable, real-time signal detectors. Since this field of research is just emerging, and some goals, such as reduced detector complexity, have to be verified via link-level performance simulations, the later steps in the design process, such as system integration, have not been investigated yet. In particular, hardware system integration and system-level verification via over-the-air testbeds can be addressed in the future. From a hardware and systems perspective, the following work might follow this thesis.

The proposed detectors should be integrated into a broader physical layer receiver architecture. The interaction with other blocks, such as channel estimation, synchronisation, decoding, etc., is important to assess system performance and hardware cost of the overall system. Further, combining multiple blocks into a larger, closer integrated receiver might allow machine learning algorithms to further optimise the physical layer and improve performance.

Models of the Analog Front End (AFE) could be integrated into the ML framework such that the non-ideal behaviour of the AFE can be compensated by the baseband processing.

Over-the-air testing of ML enhanced detectors has been limited so far. The work in [67] implements an OFDM receiver and over-the-air testing via commercial SDR radio front ends. Over-the-air testing can provide valuable information on the resilience of the trained algorithm and the possible necessity of online training during deployment.

6.4 Future Research for ML in the physical layer

The research opportunities in the physical layer are vast, and many areas might benefit from ML techniques. Examples are massive MIMO, and in particular cell-free massive MIMO, new waveforms for OFDM, non-orthogonal multiple-access techniques, reconfigurable surfaces, and more [13].

In a more general sense, we see the challenges as follows:

- The generalisation of ML models and their resilience needs to be improved. The models need to be able to handle new operating conditions (i.e. channel conditions, front-end effects, etc.) without requiring major retraining.
- As laid out throughout this thesis, the real-time constraints on physical layer blocks are challenging. Reducing deployment complexity as much as possible is an ongoing research effort in the field.

- Much research has been done on small or well-behaved systems (e.g. on non-realistic channel conditions). More research is required to make ML systems standard compliant. This also will mean being flexible to support the many configurations required by the standards.
- Opened up by ML in the physical layer, joint optimisation in an end-to-end learning process is a promising concept. As briefly mentioned before, eliminating classical block boundaries might enable the baseband ML processor to find global optima. This might include learning new waveforms, symbol alphabets, control algorithms, and more. However, feasible end-to-end architectures and their implementation, training, complexity, and performance are still open research questions.
- As mentioned above, testbed setups and over-the-air training and testing will be of major importance to truly asses the performance, advantages, and shortcomings of the ML method. It will allow the community to assess the required generalisation, compliance and flexibility properties in detail.

Although much work remains to make high-performance, scalable, and standard-compatible ML physical layer solutions a reality, the authors in [13] expect nothing short of a ML driven physical layer revolution in 6G. Of course, not everybody in the field agrees with this view; however, ML for the physical layer undoubtedly has built up momentum in the last few years. New journals like the recently launched "IEEE Transactions on Machine Learning in Communications and Networking" address this need. Many relevant conferences host tracks or special sessions on ML for communications. Also, from a standardisation perspective, in 3GPP release 18, a new study item investigating the usage of AI and ML for the air interface has been defined [47]. One may look curiously and positively towards the development of this field.

Appendix A

List of Publications

A.1 Conference Papers

1. S. Brennsteiner, T. Arslan, and J. Thompson, “Evaluation of Partially Constant, Fine-Grained, Dynamic Partial Reconfigurable Functions in FPGAs,” in 2019 International Conference on Field-Programmable Technology (ICFPT), Dec. 2019, pp. 347–350

2. S. Brennsteiner, T. Arslan, J. S. Thompson, and A. McCormick, “A machine learning enhanced approximate message passing massive MIMO accelerator,” in 2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS), Jun. 2022, pp. 443–446.

A.2 Journal Papers

1. S. Brennsteiner, T. Arslan, J. Thompson, and A. McCormick, “A Real-Time Deep Learning OFDM Receiver,” ACM Transactions on Reconfigurable Technology and Systems, vol. 15, no. 3, pp. 26:1–26:25, Dec. 2021.

2. S. Brennsteiner, T. Arslan, J. S. Thompson, and A. McCormick, “LAMA-Net: A Real-Time, Machine Learning-Enhanced Approximate Message Passing Detector for Massive MIMO,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, pp. 1–14, 2022

Appendix B

Background

This appendix provides background information on communication fundamentals and FPGA fundamentals. It might be useful for readers which are not familiar yet with one or the other. The here provided information is used throughout the thesis.

B.1 Communication Fundamentals

B.1.1 Noise and Signal Power

Many noise sources follow a normal distribution, and generally speaking, the normal distribution is observed in many random processes as explained by the central limit theorem [152, chapter 7]. The normal, or Gaussian PDF is given by

$$f(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \tag{B.1}$$

where μ and σ are the mean and standard deviation, respectively. The signal or noise variance is given by σ^2. We denote an M dimensional noise signal as $\mathcal{CN}_M(\mu, \sigma^2)$ where every entry is independent of each other and follows the PDF of Eq. (B.1).

The noise power spectral density N_0 describes the noise power per unit bandwidth. For a double-sided power spectrum of a Gaussian noise process, this is $N_0 = 2\sigma^2$. Based on this, the SNR can be defined as

$$SNR = \frac{E_S}{N_0}, \tag{B.2}$$

where E_S is the signal energy. Due to its simplicity, the AWGN noise model is often used. The noise is additive to the signal, Gaussian distributed with zero mean and has a flat power spectral density.

B.1.2 Coherence Interval

The coherence interval is a time-frequency space in which the channel in a communication system can be assumed to be constant [2, chapter 2]. The coherence interval is a consequence of multi-path propagation environments. It consists of two measures: coherence time and coherence bandwidth. The coherence time depends on the user's speed with respect to the base station. In a simple two-path propagation model, it can be estimated as

$$T_C = \frac{\lambda}{2v}[s], \tag{B.3}$$

where λ is the wavelength of the considered signal and v is the user's speed. Even though this model is simple, it typically approximates real channel models well [2, chapter 2]. In the simple two-path propagation model, the coherence bandwidth can be defined as [2, chapter 2]

$$B_C = \frac{c}{|d_1 - d_2|}[Hz], \tag{B.4}$$

where c is the speed of light and d_1 and d_2 are the path lengths. The coherence bandwidth equals the channel delay spread in this simple model with two equal amplitude multi-path components. The delay spread is defined as the difference in the time of arrival of the first and last significant multi-path components.

B.1.3 Basic Signal Estimation

Basic signal estimators such as the MMSE estimator are ubiquitous in communication systems and are the simplest methods to perform tasks like signal detection and channel estimation. This section introduces a generic system of linear equations and briefly derives the MMSE estimator for it. First, the estimator without consideration of noise is derived, which is also called the ZF in detection. In other domains such as channel estimation, the same estimator is known as the LS estimator. In later sections, the usage of the estimators for OFDM and massive MIMO signal detection is introduced. Let us assume the following generic system of linear equations

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{n}, \tag{B.5}$$

where $\boldsymbol{y} \in \mathbb{C}^M$ is a continuous random variable with the PDF $p_y(y)$ and is known. $\boldsymbol{H} \in \mathbb{C}^{MxN}$ is assumed to be fixed in value and known too. The vector $\boldsymbol{x} \in \mathbb{C}^N$ is an unknown random variable, has a PDF $p_x(x)$, and shall be estimated by the process. The noise vector is Gaussian distributed $\boldsymbol{n} \sim \mathcal{CN}_N(\mu, \sigma^2)$.

Zero Forcing (ZF)

The idea of the ZF is to find the value of $\boldsymbol{x}$ which minimises the square error between the observation $\boldsymbol{y}$ and the resulting observation from choosing $\boldsymbol{x}$:

$$\begin{aligned}
\widehat{\boldsymbol{x}} &= \arg\min_{\boldsymbol{x}} \left\{ \|\boldsymbol{y} - \boldsymbol{H}\boldsymbol{x}\|_2^2 \right\}, \\
&= \arg\min_{\boldsymbol{x}} \left\{ (\boldsymbol{y} - \boldsymbol{H}\boldsymbol{x})^H (\boldsymbol{y} - \boldsymbol{H}\boldsymbol{x}) \right\}, \\
&= \arg\min_{\boldsymbol{x}} \left\{ \boldsymbol{y}^H\boldsymbol{y} - \boldsymbol{y}^H\boldsymbol{H}\boldsymbol{x} - (\boldsymbol{H}\boldsymbol{x})^H \boldsymbol{y} + (\boldsymbol{H}\boldsymbol{x})^H\boldsymbol{H}\boldsymbol{x} \right\}, \\
&= \arg\min_{\boldsymbol{x}} \left\{ \boldsymbol{y}^H\boldsymbol{y} - 2\boldsymbol{y}^H\boldsymbol{x}\boldsymbol{H} + \boldsymbol{x}^H\boldsymbol{H}^H\boldsymbol{H}\boldsymbol{x} \right\}.
\end{aligned} \tag{B.6}$$

To find the minimum, the derivative of the above expression is taken according to the properties of matrix derivation in [153] and set to zero, such that

$$\begin{aligned} \frac{\partial(\boldsymbol{y}^H\boldsymbol{y} - 2\boldsymbol{y}^H\boldsymbol{x}\boldsymbol{H} + \boldsymbol{x}^H\boldsymbol{H}^H\boldsymbol{H}\boldsymbol{x})}{\partial\boldsymbol{x}} &= 0, \\ -2\boldsymbol{H}^H\boldsymbol{y} + 2\boldsymbol{H}^H\boldsymbol{H}\boldsymbol{x} &= 0. \end{aligned} \tag{B.7}$$

The ZF solution can therefore be expressed as

$$\widehat{\boldsymbol{x}} = (\boldsymbol{H}^H\boldsymbol{H})^{-1}\boldsymbol{H}^H\boldsymbol{y}. \tag{B.8}$$

Minimum Mean Square Error (MMSE)

The ZF solution is simple; however, it does not consider the additive noise of Eq. (B.5). It forces any interference between the elements of $\boldsymbol{x}$ to zero, potentially amplifying the noise. The MMSE estimator can trade off interference cancellation with noise power amplification. To consider the noise of the system, a slightly different optimisation problem is given. The MMSE estimator finds a matrix that minimises the mean square error between $\boldsymbol{x}$ and $\widehat{\boldsymbol{x}}$:

$$\begin{aligned} \boldsymbol{W}_{MMSE} &= \mathbb{E}\left\{\|\boldsymbol{x} - \widehat{\boldsymbol{x}}\|_2^2\right\}, \\ &= \arg\min_{\boldsymbol{W}} \mathbb{E}\left\{\|\boldsymbol{x} - \boldsymbol{W}\boldsymbol{y}\|_2^2\right\}, \\ &= \arg\min_{\boldsymbol{W}} \mathbb{E}\left\{(\boldsymbol{x} - \boldsymbol{W}\boldsymbol{y})(\boldsymbol{x} - \boldsymbol{W}\boldsymbol{y})^H\right\}, \\ &= \arg\min_{\boldsymbol{W}} \mathbb{E}\left\{\boldsymbol{x}\boldsymbol{x}^H - \boldsymbol{x}(\boldsymbol{W}\boldsymbol{y})^H - (\boldsymbol{W}\boldsymbol{y})\boldsymbol{x}^H + 2\boldsymbol{W}\boldsymbol{y}(\boldsymbol{W}\boldsymbol{y})^H\right\}, \\ &= \arg\min_{\boldsymbol{W}} \mathbb{E}\left\{\boldsymbol{x}\boldsymbol{x}^H - 2\boldsymbol{x}\boldsymbol{y}^H\boldsymbol{W}^H + 2\boldsymbol{W}\boldsymbol{y}\boldsymbol{y}^H\boldsymbol{W}^H\right\}, \end{aligned} \tag{B.9}$$

where the expectation operator for a continuous random variable is

$$\mathbb{E}\{x\} = \int_{-\infty}^{\infty} x p_x(x) dx. \tag{B.10}$$

Similar to the approach for the ZF estimator, also the derivation of the above expression is set to zero to find its minimum

$$\begin{aligned} \frac{\partial \mathbb{E}\left\{\boldsymbol{x}\boldsymbol{x}^H - 2\boldsymbol{x}\boldsymbol{y}^H\boldsymbol{W}^H + 2\boldsymbol{W}\boldsymbol{y}\boldsymbol{y}^H\boldsymbol{W}^H\right\}}{\partial \boldsymbol{W}} &= 0, \\ \mathbb{E}\left\{-2\boldsymbol{x}\boldsymbol{y}^H + 2\boldsymbol{y}\boldsymbol{y}^H\boldsymbol{W}^H\right\} &= 0. \end{aligned} \tag{B.11}$$

The $\boldsymbol{W}_{MMSE}$ matrix therefore can be expressed as

$$\boldsymbol{W}_{MMSE}^H = \mathbb{E}\left\{\boldsymbol{x}\boldsymbol{y}^H\right\}\mathbb{E}\left\{(\boldsymbol{y}\boldsymbol{y}^H)\right\}^{-1}. \tag{B.12}$$

B.2 Field Programmable Gate Arrays

FPGAs allow the implementation of custom digital logic in a configurable manner after the initial silicon production process. Xilinx Inc. (recently acquired by AMD Inc.) is one of the largest FPGA vendors. As Xilinx FPGAs are widely used, and due to the setup of this dissertation, the architecture of Xilinx FPGAs are briefly reviewed next. However, other vendors have similar product architectures and much of the presented information might also apply to them.

B.2.1 Primitives and FPGA architecture

FPGAs are versatile devices with heterogeneous hardware structures to deliver high performance for a wide spectrum of applications. The System on Chip (SoC), which is termed and marketed as an FPGA, contains typically many hardened IP blocks for high performance, relatively fixed functionality in combination with a **Programmable Logic (PL)** region. This PL region is what the term FPGA actually describes. A hardened IP is a functionality which is not implemented in the PL region but with fixed functionality during the SoC design process. This allows higher performance for certain IPs, which do not require a high degree of logic flexibility. An example is the **Processing**

System (PS) in the Xilinx Zynq series. It typically features multiple application processors and real-time processors from ARM for CPU workloads. Other examples are gigabit Ethernet transceivers, DDR memory interfaces, ADC, and DAC interfaces. Also, clocking resources such as Xilinx's mixed-mode clock manager blocks, pinning blocks, clock distribution resources and many other support structures are present to help the chip function correctly.

The PL region is tightly interconnected with the hardened IPs and consists of diverse primitives implementing the configured logic function. The most apparent primitives for implementing digital circuits are LUTs and FFs. In the Xilinx Ultrascale+ architecture, the LUTs have six inputs and two outputs. They can implement an arbitrary logic function with six inputs and one output or five inputs and two outputs per LUT. In the latter case, two inputs are shared between both logic functions. Each LUT is paired with two FF for synchronous memory. This basic structure is shown in Fig. B.1. Eight of these basic structures form so-called slices. In Ultrascale+ and many other architectures, one **CLB** contains exactly one slice. In Xilinx FPGAs, two types of slices are implemented, namely **SLICEL** and **SLICEM**. SLICEL types are intended to implement combinatorial logic via the LUT and potentially register the results in the FF for sequential slices. They also have interconnections between primitives to implement more complex logical functions. SLICEM types allow using the LUT as asynchronous memory storing up to $2^6 = 64$ bit per LUT and a total of $512\,\mathrm{bit}$ per SLICEM. The Xilinx terminology for this is the **distributed Random-Access Memory (RAM)**.

To interpret the reported utilisation results correctly, it is important to clarify commonly used terms. When results are reported as FF or LUT, this refers to the actual number of LUTs and FFs used in the design, not the number of slices, and not the number CLBs in design. In the Xilinx synthesis reports, this is mentioned as "CLB Registers" and "CLB LUTs". Xilinx reports also contain the terms "CLBM" and "CLBL", which correspond to SLICEL and SLICEM for Ultrascale+ architectures.

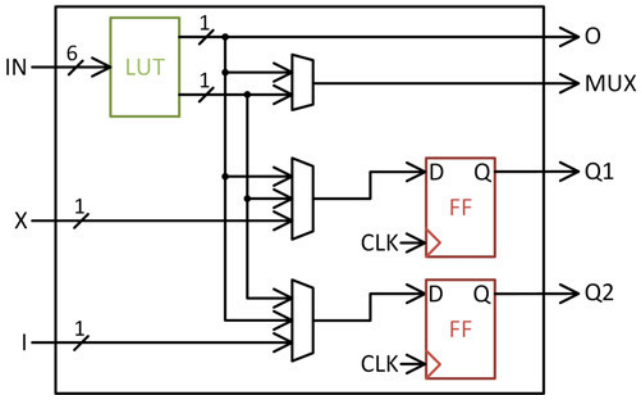


Figure B.1: Simplified, schematic depiction of a Xilinx FPGA primitives contained in a slice. Two FFs and and a LUT are the core elements [5].

There are better choices for storing large amounts of data than the distributed RAM due to its high usage of slices. For these cases, hardened memory blocks called **BRAMs** are provided. In the Ultrascale architecture (and others), each BRAM can store up to $36\,\mathrm{kbit}$ with a word-width of $36\,\mathrm{bit}$ and a depth of 1024 words. The word width describes the maximum number of bits which can be read/written per clock cycle. The word width can be up to 72 bits for a single port BRAM configuration. At first, the word width of 72 bits might seem arbitrary as it is not a power of two. However, the BRAM can be used in an error-correcting fashion in which $8\,\mathrm{bit}$ per $72\,\mathrm{bit}$ BRAM are used for automatic Hamming code error correction [154].

Even though large FPGAs have thousands of BRAMs, the total memory capacity is typically limited to a few megabytes. In addition to BRAMs, high-end Xilinx FPGAs feature, large on-chip memory blocks named **URAM**. One URAM has $288\,\mathrm{kbit}$ of synchronous storage with a $72\,\mathrm{bit}$ wide interface [154]. The number of URAM blocks in FPGAs is typically much lower than the number of BRAMs.

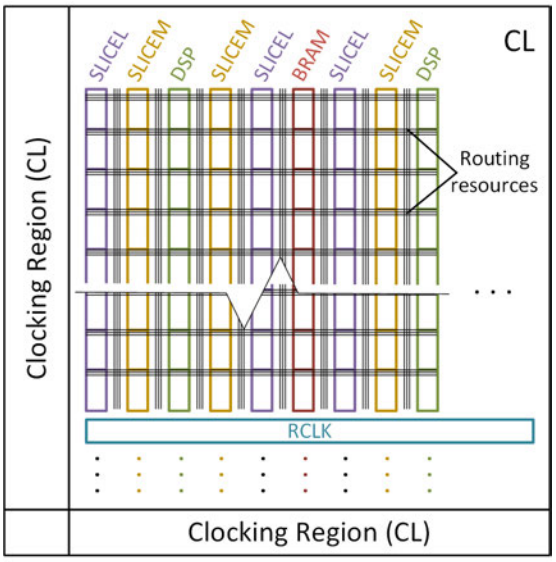


Figure B.2: Organisation of the Xilinx Ultrascale+ PL into clocking regions with heterogeneous columns of primitives.

Apart from memory and logic primitives, FPGAs typically also feature some hardened **DSP primitives**. As the name suggests, these primitives allow for efficient implementation of Digital Signal Processing (DSP) on the chip. In the Xilinx Ultrascale architecture these slices feature a $27\,\text{bit}$ pre-adder, a 27×18 bit multiplier and a $48\,\text{bit}$ Arithmetic Logic Unit (ALU) [155]. Registers for synchronous storage and other features are also part of the configurable DSP slices.

In Xilinx architectures, the primitives are organised in columns as seen in Fig. B.2. Each column contains a different number of slices, depending on the type of the slice. Columns are grouped in clocking regions. Each clocking region has independent clocking resources, thereby representing the granularity by which different clock sources are supported by the FPGA. The RCLK box groups these resources and is shown in blue in Fig. B.2.

For the primitives to implement the desired logic function, they need to be connected via wiring in a specific way. FPGAs provide for this purpose many routing resources as schematically depicted in grey in Fig. B.2. Routing resources include:

- localised, fairly fixed connections such as carry lines across columns

- flexible, but local routing resources connecting arbitrary slices in the clock region
- global routing resources, connecting between clocking regions

Abstracted for the user, the routing resources feature many programmable interconnections which allow the routing to take place. However, from a resource and latency perspective, routing is often the limiting factor in modern FPGAs.

The configuration of the FPGA defines its functionality and is stored in on-chip Static RAM (SRAM). It defines all the functionality of the primitives, including any initial values for memories and all the switches and connections of the routing resources. In short, it completely determines the functionality of the implemented logic in the FPGA. Since the SRAM memory is volatile, the configuration has to be loaded upon boot or during debugging. This configuration is also called a bitstream or configuration stream because it is typically sequentially written to the chip. The process of transforming an algorithmic concept into a bit-steam is described next.

B.2.2 Design flow

Design flows for FPGAs are complex and multifaceted. An overview of the Xilinx design flow from a system development perspective is shown in Fig. B.3. The design starts with describing the targeted logic functionality in one of many ways. The conventional way is to describe it in RTL files. However, other, more abstract ways are also available. **HLS** allows the description of the functionality via C or C++ code. The Xilinx Model composer allows using Mathworks' Matlab or Simulink software for description. These highly abstracted approaches allow for a faster and easier description of digital circuit functionality; however, they typically do not achieve as high performance as RTL code. Further, reproducibility and verification are challenges in these approaches. Each digital circuit is verified via digital simulation to ensure functional correctness regardless of the initial description. Then, the circuit is synthesised to FPGA primitives to ensure resource usage targets are reached before finally being packaged as an IP to be used in the next step.

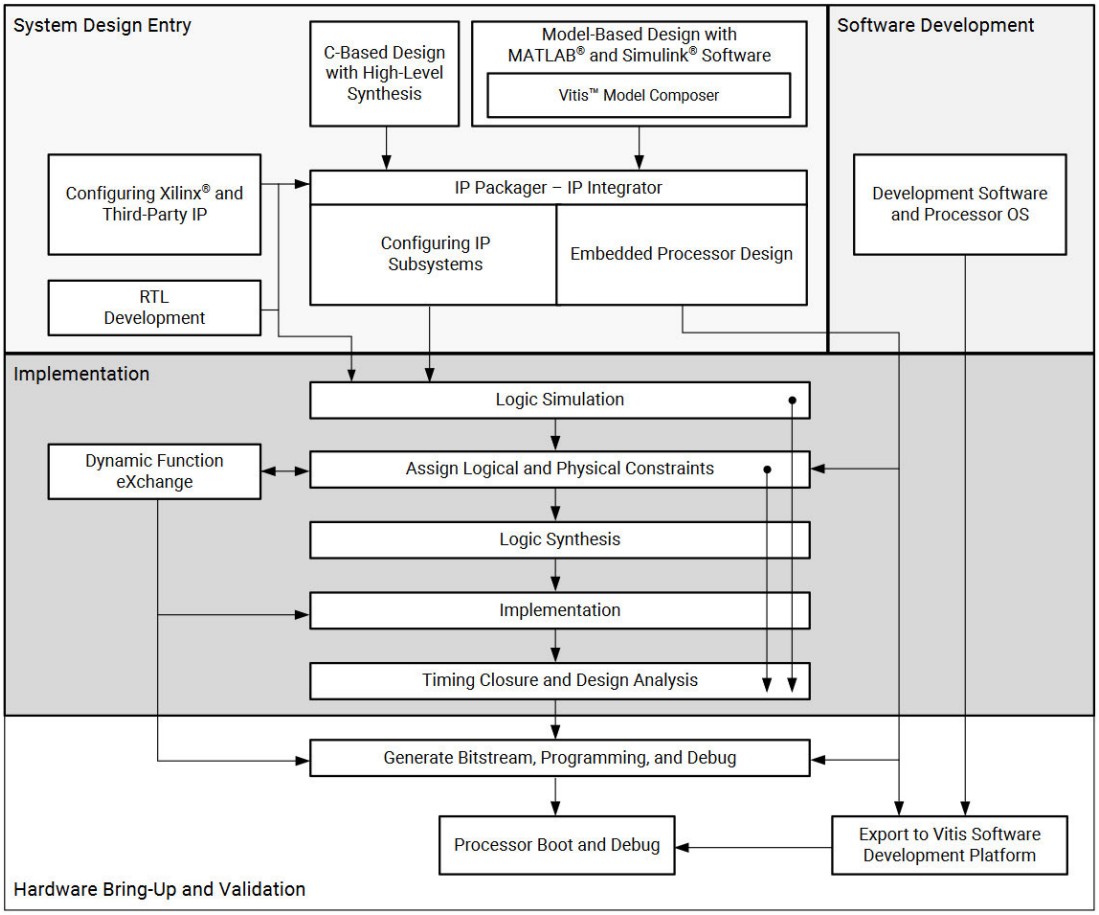


Figure B.3: Overview of the system design flow in Xilinx FPGAs [6, Figure 1]

Once all IPs are ready, the **top-level design flow** starts by instantiating the IPs and connecting them appropriately. Then, top-level verification via logic simulation is performed, followed by logic synthesis. At this point, the design is a netlist containing wiring and FPGA primitives only. This is transformed via the implementation process, which maps the netlist onto physical FPGA resources and provides the wiring between the primitives. This process is also called "Implementation" in Xilinx terminology or, more commonly, P&R. Next, the Static Timing Analysis (STA) and Design Rule Check (DRC) are performed to ensure the resulting circuit meets the FPGAs design criteria and is viable to be deployed. Implementing and ensuring the circuit viability is often iterative, complex, and time-consuming for large designs. Once the design passes this stage, a bitstream can be generated, and the FPGA can be configured.

The PL of FPGAs is typically embedded into a larger system running some software. In the embedded deployment case, some tightly coupled processors often execute application software. In data-centre applications, a host processor controls the FPGA accelerator card. Regardless of the deployment scenario, the design flow also considers **software development**. With Xilinx devices, the Vitis unified software development platform enables this. First, a platform (describing the FPGA design) is provided and then, the software, including operating systems, libraries, and application code, is developed.

During hardware **bring-up and validation**, the bitstream is deployed, and the software is loaded accordingly. The system is now running and can be validated to ensure functional correctness.

B.2.3 Fixed Point Numbers and Arithmetic

As computers are binary machines with limited resources, the quantisation of continuous numbers needs to be considered. A common way is to implement floating-point arithmetic according to the Institute of Electrical and Electronics Engineers (IEEE) 754 standard [87]. This format consists of a sign bit (S), an exponent (E), and the mantissa (M). The represented number is defined as $d = (-1)^S \times b^E \times M$, where S can have the value of 0 or 1 and b is the

radix of numerical value 2 or 10. This notation allows covering a large range of numbers thanks to the exponent. The word length is defined as 16, 32, 64, or 128 bits. However, floating-point computation is expensive when compared to fixed-point representation. This is also why the DSP blocks in Xilinx FPGAs do not support floating-point arithmetic. Other vendors, such as Intel, take a different trade-off in their high-end Arria 10 and Stratix 10 FPGAs and SoCs by supporting the IEEE 754 standard.

Since this thesis evaluates the designs on Xilinx FPGAs, fixed-point arithmetic is used as much as possible. The signed, twos-compliment fixed-point number format is specified by three components: a signed bit (S), an integer part (I), and a fractional part (F). The represented number is $d = S \times -(2^{MSB}) + I + 2^{-FL} \times F$, where FL is the number of fractional bits and I and F are binary words. A schematic of the fixed point number format is shown in Fig. B.4.

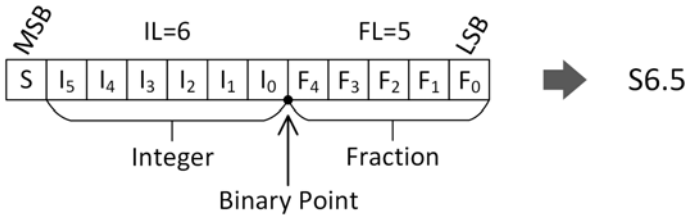


Figure B.4: Fixed-point number format illustration and example of notation

Bibliography

[1] Y. S. Cho, J. Kim, W. Y. Yang, and C. G. Kang, *MIMO-OFDM Wireless Communications with MATLAB*. John Wiley & Sons, Aug. 2010, google-Books-ID: 6HwAoeuMr3kC.

[2] T. L. Marzetta, E. G. Larsson, H. Yang, and H. Q. Ngo, *Fundamentals of Massive MIMO*. Cambridge: Cambridge University Press, 2016. [Online]. Available: https://www.cambridge.org/core/books/fundamentals-of-massive-mimo/C43AF993A6DA7075EC5F186F6BAC914B

[3] L. V. d. Perre, L. Liu, and E. G. Larsson, "Efficient DSP and Circuit Architectures for Massive MIMO: State of the Art and Future Directions," 2018. [Online]. Available: /paper/Efficient-DSP-and-Circuit-Architectures-for-Massive-Perre-Liu/9271c43ce30574f59de08fba29d47dc5087a7928

[4] K. Li, C. Jeon, J. R. Cavallaro, and C. Studer, "Decentralized equalization for massive MU-MIMO on FPGA," in *2017 51st Asilomar Conference on Signals, Systems, and Computers*, Oct. 2017, pp. 1532–1536.

[5] X. Inc., "UltraScale Architecture Configurable Logic Block User Guide (UG574)," p. 58, 2017.

[6] ——, "Vivado Design Suite User Guide Design Flows Overview (UG892)," p. 96, Apr. 2022.

[7] A. D. Ltd., "ADM-XRC-9R1: FPGA COTS board: Xilinx Zynq RFSoC - 8x 14-bit 4/5Gsps ADCs, 8x 14-bit 6.5/10Gsps DACs." Sep. 2021. [Online]. Available: https://www.alpha-data.com/esp/products.php?product=adm-xrc-9r1

[8] "Ericsson Mobility Report June 2022," p. 40, 2022.

[9] C. Zhang, P. Patras, and H. Haddadi, "Deep Learning in Mobile and Wireless Networking: A Survey," *arXiv:1803.04311 [cs]*, Jan. 2019, arXiv: 1803.04311. [Online]. Available: http://arxiv.org/abs/1803.04311

[10] V. P. Rekkas, S. Sotiroudis, P. Sarigiannidis, S. Wan, G. K. Karagiannidis, and S. K. Goudos, "Machine Learning in Beyond 5G/6G Networks—State-of-the-Art and Future Trends," *Electronics*, vol. 10, no. 22, p. 2786, Jan. 2021, number: 22 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/2079-9292/10/22/2786

[11] W. Jiang, B. Han, M. A. Habibi, and H. D. Schotten, "The Road Towards 6G: A Comprehensive Survey," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 334–366, 2021, conference Name: IEEE Open Journal of the Communications Society.

[12] K. B. Letaief, W. Chen, Y. Shi, J. Zhang, and Y.-J. A. Zhang, "The Roadmap to 6G: AI Empowered Wireless Networks," *IEEE Communications Magazine*, vol. 57, no. 8, pp. 84–90, Aug. 2019, conference Name: IEEE Communications Magazine.

[13] B. Ozpoyraz, A. T. Dogukan, Y. Gevez, U. Altun, and E. Basar, "Deep Learning-Aided 6G Wireless Networks: A Comprehensive Survey of Revolutionary PHY Architectures," *IEEE Open Journal of the Communications Society*, vol. 3, pp. 1749–1809, 2022, conference Name: IEEE Open Journal of the Communications Society.

[14] T. Wang, C.-K. Wen, H. Wang, F. Gao, T. Jiang, and S. Jin, "Deep learning for wireless physical layer: Opportunities and challenges," *China Communications*, vol. 14, no. 11, pp. 92–111, Nov. 2017. [Online]. Available: http://ieeexplore.ieee.org/document/8233654/

[15] R. G. Gallager, *Principles of Digital Communication*. Cambridge: Cambridge University Press, 2008. [Online]. Available: https://www.cambridge.org/core/books/principles-of-digital-communication/8EC09E974B567D5A045A4438759DE077

[16] H. Ye, G. Y. Li, and B.-H. Juang, "Power of Deep Learning for Channel Estimation and Signal Detection in OFDM Systems," *IEEE Wireless Communications Letters*, vol. 7, no. 1, pp. 114–117, Feb. 2018.

[17] E. Balevi and J. G. Andrews, "One-Bit OFDM Receivers via Deep Learning," *IEEE Transactions on Communications*, vol. 67, no. 6, pp. 4326–4336, Jun. 2019, conference Name: IEEE Transactions on Communications.

[18] P. A. Laplante, *Real-time systems design and analysis: tools for the practitioner*, 4th ed. Hoboken, N.J.: Wiley-IEEE Press, 2012. [Online]. Available: http://www.ezproxy.is.ed.ac.uk/login?url=http://onlinelibrary.wiley.com/book/10.1002/9781118136607

[19] S. Sesia, I. Toufik, and M. Baker, *LTE - The UMTS Long Term Evolution: From Theory to Practice*. John Wiley & Sons, Jul. 2011, google-Books-ID: beIaPXLzYKcC.

[20] 3GPP, "3GPP TR 21.915," Tech. Rep. 3GPP TR 21.915 V15.0.0 (2019-09), Sep. 2019. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3389

[21] X. Tan, W. Xu, K. Sun, Y. Xu, Y. Be'ery, X. You, and C. Zhang, "Improving Massive MIMO Message Passing Detectors With Deep Neural Network," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 2, pp. 1267–1280, Feb. 2020, conference Name: IEEE Transactions on Vehicular Technology.

[22] S. Dörner, S. Cammerer, J. Hoydis, and S. t. Brink, "Deep Learning Based Communication Over the Air," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 132–143, Feb. 2018.

[23] Y. Liang, L. Li, Y. Yi, and L. Liu, "Real-time Machine Learning for Symbol Detection in MIMO-OFDM Systems," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, May 2022, pp. 2068–2077, iSSN: 2641-9874.

[24] X. Gao, S. Jin, C.-K. Wen, and G. Y. Li, "ComNet: Combination of Deep Learning and Expert Knowledge in OFDM Receivers," *IEEE Communications Letters*, vol. 22, no. 12, pp. 2627–2630, Dec. 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8509622/

[25] N. Samuel, T. Diskin, and A. Wiesel, "Deep MIMO Detection," *arXiv:1706.01151 [cs, math, stat]*, Jun. 2017, arXiv: 1706.01151. [Online]. Available: http://arxiv.org/abs/1706.01151

[26] M. Khani, M. Alizadeh, J. Hoydis, and P. Fleming, "Adaptive Neural Signal Detection for Massive MIMO," *arXiv:1906.04610 [cs, eess, stat]*, Jun. 2019, arXiv: 1906.04610. [Online]. Available: http://arxiv.org/abs/1906.04610

[27] S. Brennsteiner, T. Arslan, J. Thompson, and A. McCormick, "A Real-Time Deep Learning OFDM Receiver," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 15, no. 3, pp. 26:1–26:25, Dec. 2021. [Online]. Available: https://doi.org/10.1145/3494049

[28] S. Brennsteiner, T. Arslan, J. S. Thompson, and A. McCormick, "A machine learning enhanced approximate message passing massive MIMO accelerator," in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, Jun. 2022, pp. 443–446.

[29] ——, "LAMANet: A Real-Time, Machine Learning-Enhanced Approximate Message Passing Detector for Massive MIMO," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 1–14, 2022.

[30] S. Brennsteiner, T. Arslan, and J. Thompson, "Evaluation of Partially Constant, Fine-Grained, Dynamic Partial Reconfigurable Functions in FPGAs," in *2019 International Conference on Field-Programmable Technology (ICFPT)*, Dec. 2019, pp. 347–350, iSSN: null.

[31] H. He, C.-K. Wen, S. Jin, and G. Y. Li, "A Model-Driven Deep Learning Network for MIMO Detection," p. 5, Sep. 2018.

[32] E. Dahlman, S. Parkvall, and J. Sköld, *5G NR: the next generation wireless access technology*, second edition ed. Amsterdam: Academic Press, 2021, oCLC: 1198592833. [Online]. Available: http://public.eblib.com/choice/PublicFullRecord.aspx?p=6353377

[33] ITU-R, "IMT Vision – Framework and overall objectives of the future development of IMT for 2020 and beyond," *M.2083-0*, p. 21, Sep. 2015.

[34] M. Shafi, A. F. Molisch, P. J. Smith, T. Haustein, P. Zhu, P. D. Silva, F. Tufvesson, A. Benjebbour, and G. Wunder, "5G: A Tutorial Overview of Standards, Trials, Challenges, Deployment, and Practice," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 6, pp. 1201–1221, Jun. 2017.

[35] 3GPP, "TS 38.211, Release 17," Tech. Rep., Dec. 2021. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3213

[36] L. Zhang, Y.-C. Liang, and D. Niyato, "6G Visions: Mobile ultra-broadband, super internet-of-things, and artificial intelligence," *China Communications*, vol. 16, no. 8, pp. 1–14, Aug. 2019, conference Name: China Communications.

[37] A. Goldsmith, *Wireless Communications*. Cambridge: Cambridge University Press, 2005. [Online]. Available: https://www.cambridge.org/core/books/wireless-communications/800BA8A8211FBECB133A7BB77CD2E2BD

[38] H. Lin, "Flexible Configured OFDM for 5G Air Interface," *IEEE Access*, vol. 3, pp. 1861–1870, 2015, conference Name: IEEE Access.

[39] M. B. Sutar and V. S. Patil, "LS and MMSE estimation with different fading channels for OFDM system," in *2017 International conference of Electronics, Communication and Aerospace Technology (ICECA)*, vol. 1, Apr. 2017, pp. 740–745.

[40] S. Coleri, M. Ergen, A. Puri, and A. Bahai, "Channel estimation techniques based on pilot arrangement in OFDM systems," *IEEE Transactions on Broadcasting*, vol. 48, no. 3, pp. 223–229, Sep. 2002, conference Name: IEEE Transactions on Broadcasting.

[41] T.-D. Chiueh and P.-Y. Tsai, *OFDM Baseband Receiver Design for Wireless Communications*. John Wiley & Sons, Apr. 2008, google-Books-ID: vhKkLimLrQoC.

[42] 3GPP, "3GPP TR 38.101, Release 17.7.0," Tech. Rep., May 2022. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3283

[43] S. A. Khwandah, J. P. Cosmas, P. I. Lazaridis, Z. D. Zaharis, and I. P. Chochliouros, "Massive MIMO Systems for 5G Communications," *Wireless Personal Communications*, vol. 120, no. 3, pp. 2101–2115, Oct. 2021. [Online]. Available: https://doi.org/10.1007/s11277-021-08550-9

[44] D. Mi, M. Dianati, L. Zhang, S. Muhaidat, and R. Tafazolli, "Massive MIMO Performance With Imperfect Channel Reciprocity and Channel Estimation Error," *IEEE Transactions on Communications*, vol. 65, no. 9, pp. 3734–3749, Sep. 2017, conference Name: IEEE Transactions on Communications.

[45] J. Tan and L. Dai, "Channel Feedback in TDD Massive MIMO Systems With Partial Reciprocity," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 12, pp. 12 960–12 974, Dec. 2021, conference Name: IEEE Transactions on Vehicular Technology.

[46] H. Jin, K. Liu, G. Lee, E. J. Farag, M. Zhang, D. Zhu, L. Zhang, E. Onggosanusi, M. Shafi, and H. Tataria, "Massive MIMO Evolution Towards 3GPP Release 18," Oct. 2022, arXiv:2210.08218 [cs, eess, math]. [Online]. Available: http://arxiv.org/abs/2210.08218

[47] X. Lin, “An Overview of 5G Advanced Evolution in 3GPP Release 18,” *IEEE Communications Standards Magazine*, vol. 6, no. 3, pp. 77–83, Sep. 2022, conference Name: IEEE Communications Standards Magazine.

[48] L. Liu, G. Peng, and S. Wei, *Massive MIMO Detection Algorithm and VLSI Architecture*. Springer, Feb. 2019, google-Books-ID: JheJDwAAQBAJ.

[49] H. Q. Ngo, Linköpings universitet, and Institutionen för systemteknik, “Massive MIMO: Fundamentals and System Designs,” Ph.D. dissertation, Department of Electrical Engineering, Linköping University, Linköping, 2015, oCLC: 942243696.

[50] M. A. Albreem, M. Juntti, and S. Shahabuddin, “Massive MIMO Detection Techniques: A Survey,” *IEEE Communications Surveys Tutorials*, vol. 21, no. 4, pp. 3109–3132, 2019, conference Name: IEEE Communications Surveys Tutorials.

[51] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, ser. Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2016.

[52] B. C. Csáji and others, “Approximation with artificial neural networks,” *Faculty of Sciences, Etvs Lornd University, Hungary*, vol. 24, no. 48, p. 7, 2001, publisher: Citeseer.

[53] T. O’Shea and J. Hoydis, “An Introduction to Deep Learning for the Physical Layer,” *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 563–575, Dec. 2017. [Online]. Available: http://ieeexplore.ieee.org/document/8054694/

[54] H. He, C.-K. Wen, S. Jin, and G. Y. Li, “Model-Driven Deep Learning for Joint MIMO Channel Estimation and Signal Detection,” *arXiv:1907.09439 [cs, eess, math]*, Jul. 2019, arXiv: 1907.09439 version: 1. [Online]. Available: http://arxiv.org/abs/1907.09439

[55] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” Jan. 2017, arXiv:1412.6980 [cs]. [Online]. Available: http://arxiv.org/abs/1412.6980

[56] C. C. Aggarwal, *Neural Networks and Deep Learning: A Textbook*. Cham: Springer International Publishing, 2018. [Online]. Available: http://link.springer.com/10.1007/978-3-319-94463-0

[57] D. Maaz, A. Galindo-Serrano, and S. E. Elayoubi, “URLLC User Plane Latency Performance in New Radio,” in *2018 25th International Conference on Telecommunications (ICT)*, Jun. 2018, pp. 225–229.

[58] Nvidia, “Nvidia A100 | Tensor Core GPU,” Jun. 2021. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf

[59] V. Volkov, *Understanding latency hiding on GPUs*. University of California, Berkeley, 2016.

[60] K. Li, B. Yin, M. Wu, J. R. Cavallaro, and C. Studer, “Accelerating massive MIMO uplink detection on GPU for SDR systems,” in *2015 IEEE Dallas Circuits and Systems Conference (DCAS)*, Oct. 2015, pp. 1–4.

[61] A. Boutros, E. Nurvitadhi, R. Ma, S. Gribok, Z. Zhao, J. C. Hoe, V. Betz, and M. Langhammer, “Beyond Peak Performance: Comparing the Real Performance of AI-Optimized FPGAs and GPUs,” in *2020 International Conference on Field-Programmable Technology (ICFPT)*, 2020, pp. 10–19.

[62] X. Inc., “Zynq UltraScale+ RFSoC Data Sheet: Overview (DS889),” Jan. 2022. [Online]. Available: https://docs.xilinx.com/api/khub/documents/XxludoUyYreGoBkgk2Zr3A/content?Ft-Calling-App=ft%2Fturnkey-portal&Ft-Calling-App-Version=4.0.25&filename=ds889-zynq-usp-rfsoc-overview.pdf

[63] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "[DL] A Survey of FPGA-based Neural Network Inference Accelerators," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 12, no. 1, pp. 1–26, Mar. 2019. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3310278.3289185

[64] A. Shawahna, S. M. Sait, and A. El-Maleh, "FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review," *IEEE Access*, vol. 7, pp. 7823–7859, 2019.

[65] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," *arXiv:1510.00149 [cs]*, Oct. 2015, arXiv: 1510.00149. [Online]. Available: http://arxiv.org/abs/1510.00149

[66] A. Felix, S. Cammerer, S. Dorner, J. Hoydis, and S. Ten Brink, "OFDM-Autoencoder for End-to-End Learning of Communications Systems," in *2018 IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. Kalamata: IEEE, Jun. 2018, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/8445920/

[67] P. Jiang, T. Wang, B. Han, X. Gao, J. Zhang, C.-K. Wen, S. Jin, and G. Y. Li, "AI-Aided Online Adaptive OFDM Receiver: Design and Experimental Results," *IEEE Transactions on Wireless Communications*, vol. 20, no. 11, pp. 7655–7668, Nov. 2021, conference Name: IEEE Transactions on Wireless Communications.

[68] Z. Zhao, M. C. Vuran, F. Guo, and S. D. Scott, "Deep-Waveform: A Learned OFDM Receiver Based on Deep Complex-Valued Convolutional Networks," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 8, pp. 2407–2420, Aug. 2021, conference Name: IEEE Journal on Selected Areas in Communications.

[69] M. Yao, M. Sohul, V. Marojevic, and J. H. Reed, "Artificial Intelligence Defined 5G Radio Access Networks," *IEEE Communications Magazine*, vol. 57, no. 3, pp. 14–20, Mar. 2019.

[70] Q. Mao, F. Hu, and Q. Hao, "Deep Learning for Intelligent Wireless Networks: A Comprehensive Survey," *IEEE Communications Surveys Tutorials*, vol. 20, no. 4, pp. 2595–2621, 2018.

[71] A. F. Molisch, *Wireless Communications*. John Wiley & Sons, Feb. 2012, google-Books-ID: 877tFGeQo5oC.

[72] haoyye, "Github Software https://github.com/haoyye/OFDM_dnn," Mar. 2020, original-date: 2017-10-09T02:14:52Z. [Online]. Available: https://github.com/haoyye/OFDM_DNN

[73] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," *arXiv:1712.05877 [cs, stat]*, Dec. 2017, arXiv: 1712.05877. [Online]. Available: http://arxiv.org/abs/1712.05877

[74] "google-research/google-research," library Catalog: github.com. [Online]. Available: https://github.com/google-research/google-research

[75] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *FPGA*, 2015.

[76] X. Inc., "DPU for Convolutional Neural Network v3.0, DPU IP Product Guide," p. 54, 2019.

[77] T. Posewsky and D. Ziener, "Throughput Optimizations for FPGA-based Deep Neural Network Inference," *Microprocessors and Microsystems*, vol. 60, pp. 151–161, Jul. 2018, arXiv: 1810.00722. [Online]. Available: http://arxiv.org/abs/1810.00722

[78] H. Zeng, R. Chen, C. Zhang, and V. Prasanna, "A Framework for Generating High Throughput CNN Implementations on FPGAs," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '18. New York, NY, USA: Association for Computing Machinery, Feb. 2018, pp. 117–126. [Online]. Available: https://doi.org/10.1145/3174243.3174265

[79] X. Inc., "UltraScale Architecture-Based FPGAs Memory IP v1.4 LogiCORE IP Product Guide," 2020.

[80] J. Van Leeuwen, "On the Construction of Huffman Trees." Jan. 1976, pp. 382–410.

[81] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," *arXiv:1602.01528 [cs]*, Feb. 2016, arXiv: 1602.01528. [Online]. Available: http://arxiv.org/abs/1602.01528

[82] C. Pal, S. Pankaj, W. Akram, A. Acharyya, and D. Biswas, "Modified Huffman based compression methodology for Deep Neural Network Implementation on Resource Constrained Mobile Platforms," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2018, pp. 1–5, iSSN: 2379-447X.

[83] M. Ledwon, B. F. Cockburn, and J. Han, "High-Throughput FPGA-Based Hardware Accelerators for Deflate Compression and Decompression Using High-Level Synthesis," *IEEE Access*, vol. 8, pp. 62 207–62 217, 2020, conference Name: IEEE Access.

[84] M. S. Abdelfattah, A. Hagiescu, and D. Singh, "Gzip on a chip: high performance lossless data compression on FPGAs using OpenCL," in *Proceedings of the International Workshop on OpenCL 2013 & 2014 - IWOCL '14*. Bristol, United Kingdom: ACM Press, 2014, pp. 1–9. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2664666.2664670

[85] Z. Aspar, Z. Mohd Yusof, and I. Suleiman, "Parallel Huffman decoder with an optimized look up table option on FPGA," in *2000 TENCON Proceedings. Intelligent Systems and Technologies for the New Millennium (Cat. No.00CH37119)*, vol. 1, Sep. 2000, pp. 73–76 vol.1.

[86] T. Posewsky and D. Ziener, “Efficient deep neural network acceleration through FPGA-based batch processing,” in *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Nov. 2016, pp. 1–8.

[87] “IEEE Standard for Floating-Point Arithmetic,” *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, Jul. 2019, conference Name: IEEE Std 754-2019 (Revision of IEEE 754-2008).

[88] Y. Ma, N. Suda, Y. Cao, S. Vrudhula, and J.-s. Seo, “ALAMO: FPGA acceleration of deep learning algorithms with a modularized RTL compiler,” *Integration, the VLSI Journal*, vol. 62, pp. 14–23, Jun. 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167926017304777

[89] J. Qiu, S. Song, Y. Wang, H. Yang, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, and N. Xu, “Going Deeper with Embedded FPGA Platform for Convolutional Neural Network,” in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA ’16*. Monterey, California, USA: ACM Press, 2016, pp. 26–35. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2847263.2847265

[90] U. Aydonat, S. O’Connell, D. Capalija, A. C. Ling, and G. R. Chiu, “An OpenCL Deep Learning Accelerator on Arria 10,” in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA ’17. New York, NY, USA: Association for Computing Machinery, Feb. 2017, pp. 55–64. [Online]. Available: https://doi.org/10.1145/3020078.3021738

[91] K. ElWazeer, M. M. Khairy, H. A. H. Fahmy, and S. E. D. Habib, “FPGA implementation of an improved channel estimation algorithm for mobile WiMAX,” in *2009 International Conference on Microelectronics - ICM*, Dec. 2009, pp. 280–283, iSSN: 2159-1679.

[92] T.-D. Chiueh and P.-Y. Tsai, *OFDM baseband receiver design for wireless communications*. Singapore ; Hoboken, NJ: John Wiley and Sons (Asia), 2007, oCLC: ocn137222764.

[93] M. U. Kingdom, "Estimate channel using input data and reference subcarriers," Sep. 2021. [Online]. Available: https://uk.mathworks.com/help/wireless-hdl/ref/ofdmchannelestimator.html

[94] ——, "Equalize OFDM data using channel estimates," Sep. 2021. [Online]. Available: https://uk.mathworks.com/help/wireless-hdl/ref/ofdmequalizer.html

[95] 3GPP, "3GPP TR 36.912," Tech. Rep. 3GPP TR 36.912 V16.0.0 (2020-07-14), Jul. 2020. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2584

[96] ——, "Technical Specification 36.211 - Physical channels and modulation (Release 16)," Tech. Rep. 3GPP TR 36.211 V16.6.0 (2021-06-30), Jun. 2021. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2425

[97] T. J. O'Shea, T. Erpek, and T. C. Clancy, "Deep Learning Based MIMO Communications," *arXiv:1707.07980 [cs, math]*, Jul. 2017, arXiv: 1707.07980. [Online]. Available: http://arxiv.org/abs/1707.07980

[98] J. Tao, J. Chen, J. Xing, S. Fu, and J. Xie, "Autoencoder Neural Network Based Intelligent Hybrid Beamforming Design for mmWave Massive MIMO Systems," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 3, pp. 1019–1030, Sep. 2020, conference Name: IEEE Transactions on Cognitive Communications and Networking.

[99] B. Wang, K. Xu, S. Zheng, H. Zhou, and Y. Liu, "A Deep Learning-Based Intelligent Receiver for Improving the Reliability of the MIMO Wireless Communication System," *IEEE Transactions on Reliability*, vol. 71, no. 2, pp. 1104–1115, Jun. 2022, conference Name: IEEE Transactions on Reliability.

[100] M.-S. Baek, S. Kwak, J.-Y. Jung, H. M. Kim, and D.-J. Choi, "Implementation Methodologies of Deep Learning-Based Signal Detection for Conventional MIMO Transmitters," *IEEE Transactions on Broadcasting*, vol. 65, no. 3, pp. 636–642, Sep. 2019, conference Name: IEEE Transactions on Broadcasting.

[101] Q. Chen, S. Zhang, S. Xu, and S. Cao, "Efficient MIMO Detection with Imperfect Channel Knowledge - A Deep Learning Approach," in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, Apr. 2019, pp. 1–6, iSSN: 1558-2612.

[102] Z. Jia, W. Cheng, and H. Zhang, "A Partial Learning-Based Detection Scheme for Massive MIMO," *IEEE Wireless Communications Letters*, vol. 8, no. 4, pp. 1137–1140, Aug. 2019, conference Name: IEEE Wireless Communications Letters.

[103] B. Hassibi and H. Vikalo, "On the sphere-decoding algorithm I. Expected complexity," *IEEE Transactions on Signal Processing*, vol. 53, no. 8, pp. 2806–2818, Aug. 2005. [Online]. Available: http://ieeexplore.ieee.org/document/1468474/

[104] N. Samuel, T. Diskin, and A. Wiesel, "Learning to Detect," *IEEE Transactions on Signal Processing*, vol. 67, no. 10, pp. 2554–2564, May 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8642915/

[105] V. Corlay, J. J. Boutros, P. Ciblat, and L. Brunel, "Multilevel MIMO Detection with Deep Learning," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, Oct. 2018, pp. 1805–1809, iSSN: 2576-2303.

[106] G. Gao, C. Dong, and K. Niu, "Sparsely Connected Neural Network for Massive MIMO Detection," in *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, Dec. 2018, pp. 397–402.

[107] S. Takabe, M. Imanishi, T. Wadayama, and K. Hayashi, “Deep Learning-Aided Projected Gradient Detector for Massive Overloaded MIMO Channels,” in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, May 2019, pp. 1–6, iSSN: 1938-1883.

[108] S. Takabe, M. Imanishi, T. Wadayama, R. Hayakawa, and K. Hayashi, “Trainable Projected Gradient Detector for Massive Overloaded MIMO Channels: Data-Driven Tuning Approach,” *IEEE Access*, vol. 7, pp. 93 326–93 338, 2019, conference Name: IEEE Access.

[109] S. Kumar, A. Singh, and R. Mahapatra, “Deep Learning Based Massive-MIMO Decoder,” in *2019 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, Dec. 2019, pp. 1–6, iSSN: 2153-1684.

[110] X. Jin and H.-N. Kim, “Parallel Deep Learning Detection Network in the MIMO Channel,” *IEEE Communications Letters*, vol. 24, no. 1, pp. 126–130, Jan. 2020, conference Name: IEEE Communications Letters.

[111] Y. Wei, M.-M. Zhao, M. Hong, M.-J. Zhao, and M. Lei, “Learned Conjugate Gradient Descent Network for Massive MIMO Detection,” *IEEE Transactions on Signal Processing*, vol. 68, pp. 6336–6349, 2020, conference Name: IEEE Transactions on Signal Processing.

[112] N. T. Nguyen and K. Lee, “Deep Learning-Aided Tabu Search Detection for Large MIMO Systems,” *IEEE Transactions on Wireless Communications*, vol. 19, no. 6, pp. 4262–4275, Jun. 2020, conference Name: IEEE Transactions on Wireless Communications.

[113] A. Datta, M. T. Deo, and V. Bhatia, “Collaborative Learning based Symbol Detection in Massive MIMO,” in *2020 28th European Signal Processing Conference (EUSIPCO)*, Jan. 2021, pp. 1678–1682, iSSN: 2076-1465.

[114] D. L. Donoho, A. Maleki, and A. Montanari, "Message-passing algorithms for compressed sensing," *Proceedings of the National Academy of Sciences*, vol. 106, no. 45, pp. 18 914–18 919, Nov. 2009, publisher: National Academy of Sciences Section: Physical Sciences. [Online]. Available: https://www.pnas.org/content/106/45/18914

[115] ——, "Message passing algorithms for compressed sensing: I. motivation and construction," in *IEEE Information Theory Workshop 2010 (ITW 2010)*. Cairo, Egypt: IEEE, Jan. 2010, pp. 1–5. [Online]. Available: http://ieeexplore.ieee.org/document/5503193/

[116] O. Y. Feng, R. Venkataramanan, C. Rush, and R. J. Samworth, "A unifying tutorial on Approximate Message Passing," *arXiv:2105.02180 [cs, math, stat]*, May 2021, arXiv: 2105.02180. [Online]. Available: http://arxiv.org/abs/2105.02180

[117] C. Jeon, R. Ghods, A. Maleki, and C. Studer, "Optimality of large MIMO detection via approximate message passing," in *2015 IEEE International Symposium on Information Theory (ISIT)*, Jun. 2015, pp. 1227–1231, iSSN: 2157-8117.

[118] ——, "Optimal Data Detection in Large MIMO," *arXiv:1811.01917 [cs, eess, math]*, Nov. 2018, arXiv: 1811.01917. [Online]. Available: http://arxiv.org/abs/1811.01917

[119] C. Jeon, K. Li, J. R. Cavallaro, and C. Studer, "On the achievable rates of decentralized equalization in massive MU-MIMO systems," in *2017 IEEE International Symposium on Information Theory (ISIT)*, Jun. 2017, pp. 1102–1106.

[120] J. Ma and L. Ping, "Orthogonal AMP," *IEEE Access*, vol. 5, pp. 2020–2033, 2017, conference Name: IEEE Access.

[121] J. P. Vila and P. Schniter, "Expectation-Maximization Gaussian-Mixture Approximate Message Passing," *IEEE Transactions on Signal Processing*, vol. 61, no. 19, pp. 4658–4672, Oct. 2013, conference Name: IEEE Transactions on Signal Processing.

[122] D. Ito, S. Takabe, and T. Wadayama, "Trainable ISTA for Sparse Signal Recovery," *IEEE Transactions on Signal Processing*, vol. 67, no. 12, pp. 3113–3125, Jun. 2019, conference Name: IEEE Transactions on Signal Processing.

[123] P. Zheng, Y. Zeng, Z. Liu, and Y. Gong, "Deep Learning Based Trainable Approximate Message Passing for Massive MIMO Detection," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, Jun. 2020, pp. 1–6, iSSN: 1938-1883.

[124] M. Goutay, F. Ait Aoudia, and J. Hoydis, "Deep HyperNetwork-Based MIMO Detection," in *2020 IEEE 21st International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, May 2020, pp. 1–5, iSSN: 1948-3252.

[125] H. He, C.-K. Wen, S. Jin, and G. Y. Li, "Model-Driven Deep Learning for MIMO Detection," *IEEE Transactions on Signal Processing*, vol. 68, pp. 1702–1715, 2020, conference Name: IEEE Transactions on Signal Processing.

[126] F. Burkhardt, S. Jaeckel, E. Eberlein, and R. Prieto-Cerdeira, "QuaDRiGa: A MIMO channel model for land mobile satellite," in *The 8th European Conference on Antennas and Propagation (EuCAP 2014)*, Apr. 2014, pp. 1274–1278, iSSN: 2164-3342.

[127] Y. Gao, H. Niu, and T. Kaiser, "Massive MIMO Detection Based on Belief Propagation in Spatially Correlated Channels," in *SCC 2017; 11th International ITG Conference on Systems, Communications and Coding*, Feb. 2017, pp. 1–6.

[128] W. Fukuda, T. Abiko, T. Nishimura, T. Ohgane, Y. Ogawa, Y. Ohwatari, and Y. Kishiyama, "Low-Complexity Detection Based on Belief Propagation in a Massive MIMO System," in *2013 IEEE 77th Vehicular Technology Conference (VTC Spring)*, Jun. 2013, pp. 1–5, iSSN: 1550-2252.

[129] X. Liu and Y. Li, "Deep MIMO Detection Based on Belief Propagation," in *2018 IEEE Information Theory Workshop (ITW)*, Nov. 2018, pp. 1–5.

[130] S. Shahabuddin, M. Juntti, and C. Studer, "ADMM-based infinity norm detection for large MU-MIMO: Algorithm and VLSI architecture," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. Baltimore, MD, USA: IEEE, May 2017, pp. 1–4. [Online]. Available: http://ieeexplore.ieee.org/document/8050311/

[131] M.-W. Un, M. Shao, W.-K. Ma, and P. C. Ching, "Deep Mimo Detection Using ADMM Unfolding," in *2019 IEEE Data Science Workshop (DSW)*, Jun. 2019, pp. 333–337.

[132] I. N. Tiba, Q. Zhang, J. Jiang, and Y. Wang, "A Low-Complexity Admm-Based Massive Mimo Detectors Via Deep Neural Networks," in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Jun. 2021, pp. 4930–4934, iSSN: 2379-190X.

[133] Q. Zhang, X. Zhao, J. Wang, and Y. Wang, "Designing a QAM Signal Detector for Massive Mimo Systems via PS-ADMM Approach," in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2022, pp. 5333–5337, iSSN: 2379-190X.

[134] M. A. Albreem, A. H. Alhabbash, S. Shahabuddin, and M. Juntti, "Deep Learning for Massive MIMO Uplink Detectors," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 1, pp. 741–766, 2022, conference Name: IEEE Communications Surveys & Tutorials.

[135] K. Yu, M. Bengtsson, B. Ottersten, D. McNamara, P. Karlsson, and M. Beach, "A wideband statistical model for NLOS indoor MIMO channels," in *Vehicular Technology Conference. IEEE 55th Vehicular Technology Conference. VTC Spring 2002 (Cat. No.02CH37367)*, vol. 1, May 2002, pp. 370–374 vol.1.

[136] J. Zhang, Y. He, Y.-W. Li, C.-K. Wen, and S. Jin, "Meta Learning-Based MIMO Detectors: Design, Simulation, and Experimental Test," *IEEE Transactions on Wireless Communications*, vol. 20, no. 2, pp. 1122–1137, Feb. 2021, conference Name: IEEE Transactions on Wireless Communications.

[137] "IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pp. 1–3534, Dec. 2016, conference Name: IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012).

[138] Y. He, J. Zhang, C.-K. Wen, and S. Jin, "TurboNet: A Model-driven DNN Decoder Based on Max-Log-MAP Algorithm for Turbo Code," in *2019 IEEE VTS Asia Pacific Wireless Communications Symposium (APWCS)*, Aug. 2019, pp. 1–5.

[139] ARM, "AMBA 4 AXI4-Stream Protocol Specification," Tech. Rep., Mar. 2010. [Online]. Available: https://developer.arm.com/documentation/ihi0051/a/

[140] C. Jeon, O. Castañeda, and C. Studer, "A 354Mb/s 0.37mm^2 151mW 32-User 256-QAM Near-MAP Soft-Input Soft-Output Massive MU-MIMO Data Detector in 28nm CMOS," *arXiv:1908.03288 [cs, eess, math]*, Aug. 2019, arXiv: 1908.03288. [Online]. Available: http://arxiv.org/abs/1908.03288

[141] Xilinx, "Vitis High-Level Synthesis User Guide UG1399 (v2020.2)," Tech. Rep., 2020. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_2/ug1399-vitis-hls.pdf

[142] X. Inc., "Alveo U50 Data Center Accelerator Card Data Sheet (DS965)," Tech. Rep., Aug. 2020. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documents/data_sheets/ds965-u50.pdf

[143] R. Gangarajaiah, H. Prabhu, O. Edfors, and L. Liu, "A Cholesky decomposition based massive MIMO uplink detector with adaptive interpolation," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2017, pp. 1–4, iSSN: 2379-447X.

[144] D. Zhu, B. Li, and P. Liang, "On the matrix inversion approximation based on neumann series in massive MIMO systems," in *2015 IEEE International Conference on Communications (ICC)*, Jun. 2015, pp. 1763–1769.

[145] "Vitis Accelerated Libraries," Mar. 2022, original-date: 2019-09-23T19:13:46Z. [Online]. Available: https://github.com/Xilinx/Vitis_Libraries

[146] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, and I. Yamazaki, "The Singular Value Decomposition: Anatomy of Optimizing an Algorithm for Extreme Scale," *SIAM Review*, vol. 60, no. 4, pp. 808–865, Jan. 2018, publisher: Society for Industrial and Applied Mathematics. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/17M1117732

[147] M. Khani, "MMNet Github Repository." [Online]. Available: https://github.com/mehrdadkhani/MMNet

[148] 3GPP, "Technical Report 38.801 - Radio access architecture and interfaces (Release 14)," Tech. Rep. 3GPP TR 38.801 V14.0.0 (2017-04), Apr. 2017. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3056

[149] M. Wu, B. Yin, G. Wang, C. Dick, J. R. Cavallaro, and C. Studer, "Large-Scale MIMO Detection for 3GPP LTE: Algorithms and FPGA Implementations," *IEEE Journal of Selected Topics in Signal Processing*, vol. 8, no. 5, pp. 916–929, Oct. 2014, conference Name: IEEE Journal of Selected Topics in Signal Processing.

[150] Z. Wu, Y. Xue, X. You, and C. Zhang, “Hardware efficient detection for massive MIMO uplink with parallel Gauss-Seidel method,” in *2017 22nd International Conference on Digital Signal Processing (DSP)*, Aug. 2017, pp. 1–5, iSSN: 2165-3577.

[151] 3GPP, “3GPP TR 38.913, Release 17,” Tech. Rep., Mar. 2022. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2996

[152] D. C. Montgomery and G. C. Runger, *Applied Statistics and Probability for Engineers*. John Wiley & Sons, Mar. 2010, google-Books-ID: _f4KrEcNAfEC.

[153] K. B. Petersen and M. S. Pedersen, *The Matrix Cookbook*. Technical University of Denmark, Nov. 2012. [Online]. Available: http://www2.compute.dtu.dk/pubdb/pubs/3274-full.html

[154] X. Inc., “UltraScale Architecture Memory Resources User Guide (UG573),” p. 139, Sep. 2021.

[155] ——, “UltraScale Architecture DSP Slice User Guide (UG579),” p. 77, Aug. 2021.