Heterogeneous Resource Management for Services based on Artificial Intelligence

Thaha Mohammed
Heterogeneous Resource Management for Services based on Artificial Intelligence

Thaha Mohammed

A doctoral dissertation completed for the degree of Doctor of Science (Technology) to be defended, with the permission of the Aalto University School of Science, at a public examination held at the lecture hall T2 of the school on 31 July 2023 at 12:00.

Aalto University
School of Science
Department of Computer Science
Abstract

Machine-to-machine communication (M2M) is becoming the most significant share of wireless traffic, largely due to emerging applications in the Internet of Things (IoT) including those for smart cities and intelligent transportation systems. A large number of such applications leverage artificial intelligence (AI) through machine learning (ML) and have heterogeneous resource requirements. To this end, several novel computing and communication paradigms have been proposed, including cloud, fog, edge, and network slicing. This dissertation addresses heterogeneous resource management for AI-based services with a focus on distributed processing and IoT scenarios. Specifically, we leverage the fog and edge computing paradigms for efficient management of resources including processing, communication, and AI knowledge. First, we consider how to achieve fast and scalable deep neural network (DNN) inference involving IoT devices. Accordingly, we propose distributed techniques that collaboratively partition and offload computation under dynamic network conditions to minimize DNN inference time. Second, we develop a tool to improve the DNN inference time through fast sparse matrix-vector multiplication (SpMV), which is a major computing operation for pruned DNNs. The related data structures and algorithms are selected through a rigorous analysis of sparsity and prediction of the related performance. Next, we focus on efficient network resource utilization while providing a target service quality. In detail, we leverage slicing and Fog-RANs to improve resource utilization for generic services in 5G networks with multiple service providers. To this end, we propose a hierarchical resource scheduling mechanism named 2L-MRA to jointly allocate multiple Fog-RAN resources to network slices in two stages. Finally, we target improving the accuracy of the DNNs by developing an economic market that incentivizes different service providers to trade and combine their existing knowledge for higher model accuracy. Specifically, we devise a model based on Fisher’s market for optimal knowledge sharing through transfer learning and a weight fusion technique to merge the acquired knowledge.

Keywords Distributed inference, DNN offloading, Edge computing, Heterogeneous resources, Network economics
The journey culminating in this dissertation has been shaped by the unswerving support, guidance, and contributions of numerous individuals. First and foremost, I would like to express my appreciation and gratitude to my supervisor, Mario Di Francesco. Throughout this journey, Mario has been an exceptional mentor, guiding me through thick and thin. His unwavering support and dedication to my success have been a source of inspiration. I am forever indebted to you for your guidance and belief in my abilities. I would also like to extend my thanks to Prof. Jiasi Chen and Prof. Georgios Iosifidis, the pre-examiners of my thesis for their evaluation and insightful comments, as they have enhanced the quality of this work. Furthermore, I would like to extend my gratitude to Prof. Jiannong Cao, who graciously agreed to be my opponent for the public defense.

I would like to acknowledge the invaluable contributions and support from my co-authors and colleagues, particularly Carlee Joe-Wong, Rohit Babbar, Stephan Sigg, Behrouz Jedari, Rashid Mehmood, Aiiad Albeshri, Iyad Katib, Yuzhe Xu, Carlo; Fischione and Si Ahmed Naas. The engaging discussions and intellectual exchange we shared have broadened my perspective and enriched my research. Special thanks go to Waseem Ahmed and Rashid Mehmood for initially inspiring me to pursue academic research. I am grateful to the The Helsinki Doctoral Education Network in Information and Communications Technology (HICT) as well the Academy of Finland for funding my research.

I would also like to extend my gratitude to my friends and colleagues, such as Gazi Ilahi, Ashutosh Vaishnav, Jaisal Ashraf, Naveed Vaheed, Henry Carlson, and Keijo Lehtinen for their camaraderie, stimulating discussions and shared lunches. Thanks to the support from the staff at the CS IT department, HR, travel services, and student services for doctoral students. I would like to thank my workplace, Skylo, for providing me with the opportunity to work there. I extend my gratitude to Anand Radhakrishnan, Naveed Vaheed, and Marco Keskinen for their support in the completion of the thesis with work.

Finally, my deepest gratitude goes to my family, especially my parents,
for their unending support throughout this entire journey. Their love, encouragement, and belief in me have been my constant motivation. I would also like to thank my in-laws for their support and understanding. To all my friends across the globe, thank you for your friendship and support. Lastly, I express my heartfelt appreciation to my wife and son for their understanding, patience, and unwavering support amidst the challenges and demands of this research journey. Your presence and encouragement have been my strength, and I am eternally grateful for your love and support.

This thesis represents the culmination of years of dedicated research, and it would not have been possible without the contributions and support of all those mentioned above, as well as many others who have played a part in shaping my academic and personal growth. I am truly grateful for the opportunity to undertake this remarkable endeavor, and I am indebted to each individual who has been part of this journey.

Espoo, June 7, 2023,

Thaha Mohammed
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>1</td>
</tr>
<tr>
<td>Contents</td>
<td>3</td>
</tr>
<tr>
<td>List of Publications</td>
<td>5</td>
</tr>
<tr>
<td>Author’s Contribution</td>
<td>7</td>
</tr>
<tr>
<td>List of Figures</td>
<td>9</td>
</tr>
<tr>
<td>List of Tables</td>
<td>11</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>15</td>
</tr>
<tr>
<td>1.1 Scope and Research Questions</td>
<td>16</td>
</tr>
<tr>
<td>1.2 Methodology</td>
<td>19</td>
</tr>
<tr>
<td>1.3 Research Contributions</td>
<td>21</td>
</tr>
<tr>
<td>1.4 Structure of Dissertation</td>
<td>23</td>
</tr>
<tr>
<td>2. Distributed processing of AI models</td>
<td>25</td>
</tr>
<tr>
<td>2.1 Background and Motivation</td>
<td>25</td>
</tr>
<tr>
<td>2.2 Related Work</td>
<td>27</td>
</tr>
<tr>
<td>2.3 Scalable DNN Inference at the Edge</td>
<td>28</td>
</tr>
<tr>
<td>2.3.1 Network and Task Model</td>
<td>29</td>
</tr>
<tr>
<td>2.3.2 Computation Model</td>
<td>29</td>
</tr>
<tr>
<td>2.3.3 DNN Inference Offloading with Load Balancing</td>
<td>30</td>
</tr>
<tr>
<td>2.3.4 Distributed Algorithm for DNN Inference</td>
<td>32</td>
</tr>
<tr>
<td>2.3.5 Summary of Results</td>
<td>33</td>
</tr>
<tr>
<td>2.4 Fine-Grained Adaptive Partitioning and Offloading</td>
<td>35</td>
</tr>
<tr>
<td>2.4.1 Reference Scenario and Task Model</td>
<td>35</td>
</tr>
<tr>
<td>2.4.2 Communication Model</td>
<td>36</td>
</tr>
<tr>
<td>2.4.3 Problem Formulation</td>
<td>37</td>
</tr>
<tr>
<td>2.4.4 Adaptive Partitioning and Offloading</td>
<td>38</td>
</tr>
<tr>
<td>2.4.5 Summary of Results</td>
<td>40</td>
</tr>
<tr>
<td>2.5 Conclusion</td>
<td>42</td>
</tr>
</tbody>
</table>
List of Publications

This thesis consists of an overview and of the following publications which are referred to in the text by their Roman numerals.


Author’s Contribution

Publication I: “Distributed Assignment with Load Balancing for DNN Inference at the Edge”

Mohammed and Di Francesco came up with the original idea. Mohammed modeled the deep neural network offloading problem. Xu and Fischione designed the auction algorithm. Mohammed designed the experiments, implemented and evaluated the solution through simulation, and analyzed the results. All authors participated in writing the manuscript.

Publication II: “Distributed Inference Acceleration with Adaptive DNN Partitioning and Offloading”

Mohammed came up with the original idea, formulated the related optimization problem, developed the proposed solutions, implemented the simulator, designed the experimental plan, performed the evaluation, and analyzed the results. Babbar participated in modeling deep neural networks. Di Francesco and Joe-Wong supervised the study and helped explain the results. All authors contributed to the writing of the manuscript.

Publication III: “DIESEL: A novel deep learning-based tool for SpMV computations and solving sparse linear equation systems”

Mohammed and Mehmood came up with the original idea. Mohammed developed the proposed tool, designed the experimental plan, and performed the evaluation. Mohammed and Mehmood analyzed the results. Mehmood and Albeshri helped in writing the article. Mehmood and Katib supervised the research.
Author's Contribution


Jedari and Di Franceso identified the original research question. Mohammed and Jedari devised the model and formulated the related optimization problem. Mohammed designed the solution, designed the evaluation plan, implemented the simulator, and carried out the evaluation. Mohammed and Di Franceso analyzed the results. Jedari and Di Franceso helped write and revise the article. Di Franceso supervised the research.

Publication V: “Knowledge Sharing in AI Services: A Market-based Approach”

Mohammed and Naas identified the problem space. Mohammed formulated the economic market model, designed the related evaluation plan, and implemented the simulator for the economic market. Naas developed and evaluated the knowledge fusion technique. All co-authors participated in writing the article and analyzing the results. Sigg and Di Franceso supervised the research and revised the article.
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>DNN inference offloading at the edge. (a) Reference architecture with heterogeneous end devices and more powerful edge servers. (b) Sample DNN inference tasks $t_1$ and $t_2$ associated with different devices. Each task relies on a certain DNN architecture with multiple layers $l_i$. (c) Sample assignment involving DNN tasks and three edge servers.</td>
</tr>
<tr>
<td>2.2</td>
<td>Performance of DAMA: (a) total inference time of the different schemes as a function of the DNN tasks for VGG16, (b) improvement of DAMA in terms of the total inference time for the considered DNN benchmarks.</td>
</tr>
<tr>
<td>2.3</td>
<td>Load distribution for the considered schemes in a network with six edge servers as the number of (a) DNN layers and (b) partitions offloaded to each server.</td>
</tr>
<tr>
<td>2.4</td>
<td>(a) A DNN inference task consisting of multiple sub-tasks, corresponding to processing either a whole layer or part of it, namely, a partition. (b) Sample DNN offloading of task $d_1$ in (a) at end user $u_1$ onto three fog nodes; striped sub-tasks denote those associated with user nodes other than $u_1$. (c) Segment-based 1D partition of a single convolutional layer with redundant data from neighboring partitions. (d) Hybrid representation of a fully-connected layer: $P_1$ is a dense representation and $P_2$ is a sparse representation.</td>
</tr>
<tr>
<td>2.5</td>
<td>Performance of DINA for the considered benchmarks: (a) total execution time of the different schemes as a function of the DNN tasks for VGG16, (b) improvement of DINA against the other schemes in terms of total execution time, and (c) improvement of DINA over ECDI-L [68] in terms of total execution time.</td>
</tr>
<tr>
<td>2.6</td>
<td>Amount of computations at individual fog nodes for the considered schemes: (a) GANC, (b) RANDP, and (c) DINA-O.</td>
</tr>
</tbody>
</table>
3.1 Representations of matrix $A$ with: (a) Dense, (b) COO, (c) CSR, and (d) ELL storage schemes. ........................................ 44
3.2 Sparsity structure of four sparse matrices: (a) Freescale1, (b) nd24k, (c) Transport, and (d) tube1; and (e) their performance in GFLOPS for five different kernels. .......................... 45
3.3 Overview of DIESEL: features are extracted from a set of training matrices and used to train a model in the offline process; the trained model is used for predicting the storage format for given input and produce the corresponding SpMV results in the online process. ................................. 47
3.4 Performance profiles of the five SpMV kernels on GPUs for the selected sparsity features (a) $\bar{a}^m$, $z$, and $z^m$; (b) $z^m$ and $z$. ........................................ 49
3.5 The correlation matrix for the selected features. ....................... 50
3.6 Performance of all six SpMV kernels in GFLOPS for each of the 1,056 matrices in the dataset. ........................................ 51
3.7 The throughput in GFLOPS of all considered kernels against $z$ for each of the 1,056 matrices. ........................................ 51
3.8 The execution time of DIESEL (aggregated over all 1,056 matrices) compared with other kernels. ................................. 52
3.9 GFLOPS vs $n_r$ variance and $z$ for (a) DIA; (b) COO; (c) CSR; (d) ELL; (e) HYB; and (f) DIESEL. ................................. 53
4.1 System model of Fog-RAN slicing. ........................................ 58
4.2 An example of multi-resource allocation to two slices (one DL SP and one UL SP) in 2L-MRA. ........................................ 63
4.3 The total utility achieved as a function of (a) time, (b) number of SPs, and (c) task arrival rate. ........................................ 66
4.4 (a) Optimality gap of 2L-MRA as a function of RUs. (b) Jain’s fairness index of the considered schemes. (c) Utilization ratio by type. ........................................ 67
5.1 (a) Reference scenario: multiple providers offer AI-based services, relying on different DNN architectures. (b) Service providers sell and buy the weights associated with the DNNs they use as a form of market-based knowledge sharing based on transfer learning; (c) Acquired weights are used by buyers to update (part of) the knowledge in their DNNs so as to increase their inference accuracy. .......................... 72
5.2 (a) Total utility, (b) proportional fairness index for EEG and (c) envy freeness index of the considered schemes as a function of the number of sellers. ................................. 77
5.3 Accuracy as a function of the training epoch for knowledge fusion involving one buyer and one seller for different distributions of training data between the buyer and the seller: (a) 30% vs 70%, (b) 40% vs 60%, and (c) 50% vs 50%. .......................... 78
List of Tables

2.1 Fairness and convergence time of the different schemes for VGG16. ........................................ 35

3.1 Selected Features From Sparse Matrices .............. 48
3.2 Comparison of DIESEL with other AI-based works [146, 14, 15, 128] (based on our dataset and respective feature sets) 54
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CU</td>
<td>Control Unit</td>
</tr>
<tr>
<td>DAMA</td>
<td>Distributed Auction for Multiple Assignment</td>
</tr>
<tr>
<td>DINA</td>
<td>Distributed Inference Acceleration</td>
</tr>
<tr>
<td>DL</td>
<td>Down Link</td>
</tr>
<tr>
<td>DNN</td>
<td>Deep Neural Network</td>
</tr>
<tr>
<td>DRF</td>
<td>Dominant Resource Fairness</td>
</tr>
<tr>
<td>eMBB</td>
<td>enhanced Mobile BroadBand</td>
</tr>
<tr>
<td>GDP</td>
<td>Gross Domestic Product</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphical Processing Unit</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>M2M</td>
<td>Machine-to-Machine</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>mMTC</td>
<td>massive Machine Type Communication</td>
</tr>
<tr>
<td>MNO</td>
<td>Mobile Network Operator</td>
</tr>
<tr>
<td>RAN</td>
<td>Radio Access Networks</td>
</tr>
<tr>
<td>RU</td>
<td>Radio Unit</td>
</tr>
<tr>
<td>QoE</td>
<td>Quality of Experience</td>
</tr>
</tbody>
</table>
Abbreviations

**QoS**  Quality of Service  
**SLA**  Service Level Agreement  
**SpMV**  Sparse Matrix-Vector multiplication  
**SP**  Service Provider  
**TL**  Transfer Learning  
**TTI**  Transmission Time Interval  
**t-SNE**  t-Distributed Stochastic Neighbor Embedding  
**UE**  User Equipment  
**UL**  Up Link  
**URLLC**  Ultra Reliable Low Latency Communication
1. Introduction

Recent advances in information and communication technologies – such as the fifth generation (5G) of mobile networks – have resulted in a rapid increase in the number of Internet-connected devices, expected to reach 29.3 billion (three times the world population) at the end of 2023 [32]. More than half of the related traffic already consists of machine-to-machine (M2M) communication involving smart objects in the Internet of Things (IoT). IoT devices are inherently heterogeneous in terms of both their capabilities and operational requirements. For instance, they range from resource-constrained – in terms of available energy, memory size, and computation capabilities – mobile user equipment (UE), wireless sensors, and wearable devices to smart manufacturing systems equipped with high-end central processing units (CPUs), graphical processing units (GPUs), and special-purpose accelerators [58].

M2M communication among such heterogeneous devices is represented by emerging applications, including home automation, video surveillance, healthcare, and smart transportation [74]. A large number of these applications leverage artificial intelligence (AI), including machine learning (ML), reinforcement learning (RL), and deep learning (DL). For instance, AI-based applications in a smart home scenario include voice as well as facial recognition, environment analysis, and human behavior analysis. Training and inference of these ML models demand a large amount of processing capabilities, generally not available at resource-constrained devices. In addition, these applications have heterogeneous requirements in terms of the different types of services supported by 5G wireless networks [124]: enhanced mobile broadband (eMBB), ultra-reliable low-latency communications (URLLCs), and massive machine-type communications (mMTCs). For instance, emerging augmented reality media and applications as well as 360-degree streaming video are eMBB services that need high data bandwidth and moderate to low latency [140]. In contrast, a large number of low-power devices may require sending many small data packets at the same time. Such a pattern characterizes smart home sensors that enable water, gas, electricity and waste management in mMTC services [140].
Furthermore, mission-critical applications such as autonomous vehicles are latency sensitive and demand both ultra-reliable and low-latency communication in URLLC services [140].

Emerging applications require efficient computing and communication solutions to manage their ever-growing demands. To this end, different computing paradigms and communication paradigms—such as cloud, edge, fog, and network slicing—have been proposed to assist end devices. In particular, cloud computing enables resource-constrained devices to offload compute-intensive applications to powerful data centers. The cloud offers a significant amount of computing resources but it incurs high latency [8]. For this reason, edge computing [110] has been proposed: accordingly, computing and storage are made available at the network edge, closer to resource-constrained devices, thereby reducing the latency incurred by offloading computation. Furthermore, running ML tasks at the edge brings additional advantages, including higher energy efficiency, privacy protection, throughput improvement, and on-demand deployment. Edge computing could take place at base stations, 5G radio access networks (RANs), servers close to the device, routers, switches, or even wireless access points. Moreover, fog computing further pushes the availability of processing resources from the edge of the network to the end devices themselves [120]. Finally, network slicing is a communication paradigm that aims to run multiple logical networks as virtually independent services on a common physical infrastructure to provide a target quality [57]. Applications with diverse requirements greatly benefit from network slicing. For instance, a vehicle can be simultaneously provided a high-bandwidth slice for infotainment and an ultra-reliable slice for assisted driving.

Satisfying the requirements of emerging applications—including AI-based services running on resource-constrained IoT devices—introduces several key challenges. The first is related to the execution latency and management of computing resources to satisfy a given quality of service (QoS) or quality of experience (QoE). The second is to efficiently satisfy the heterogeneous resource demands of the applications economically while guaranteeing the service level agreements (SLAs) of service providers as well as the privacy of the stakeholders.

1.1 Scope and Research Questions

The research in this dissertation specifically addresses heterogeneous resource management for AI-based services, with a focus on distributed processing and IoT scenarios. This is achieved by leveraging fog/edge networks and optimal resource management including computing, communication, and AI knowledge. First, we consider how to achieve fast and scalable deep neural network (DNN) inference for AI computations
at resource-constrained IoT devices. We propose techniques that achieve efficient distributed offloading in edge and fog networks. Second, we develop a tool to improve the inference time of pruned DNNs by improving the performance of sparse matrix-vector multiplication (SpMV), which is a major computing operation for such DNNs. Then, we target improving the accuracy of the DNNs by developing a market that incentivizes different SPs to trade their existing knowledge. Finally, we propose a mechanism for joint multi-resource allocation to improve both network utilization and the profits of SPs delivering services in 5G through fog radio access networks (Fog-RANs) and network slicing.

This dissertation aims at answering the following research questions.

- **How to efficiently offload DNN inference from resource-constrained devices for fast collaborative execution?** AI-based services involve DNN inference running at resource-constrained embedded devices such as mobile phones, wearables, and smart objects in next-generation networks [174]. Unfortunately, the related algorithms generally require a significant amount of computational resources to achieve satisfactory results. Instead of entirely offloading DNN computation to the cloud, devices may distribute inference across the network edge, possibly including other end devices [68]. However, edge has limited resources compared to those in the cloud; therefore, inference offloading generally requires dividing the original DNN into different pieces that are then assigned to multiple servers or edge devices. Such an approach reduces the communication overhead while, at the same time, increasing the utilization of computing resources in the network. In this context, the main challenge is deciding how to collaboratively partition and distribute computation under dynamic network conditions. The related solutions in the state of the art either make strong assumptions on the system model or fail to provide strict performance guarantees, especially in terms of efficient and fair use of resources. In contrast, our goal is to devise distributed solutions that solve the partitioning and offloading problem in polynomial time with provable guarantees.

- **How to employ hardware acceleration to speedup inference involving pruned DNNs characterized by different sparsity patterns?** Sparse matrix-vector multiplication (SpMV) is a major operation to carry out inference with pruned DNNs, which have been “compressed” by appropriately setting weights to zero [152]. Optimizing SpMV computation on hardware accelerators improves the overall pruned DNN inference time. The sparsity pattern of the matrix determines the better-performing sparse storage scheme and the related SpMV algorithm. However, sparsity patterns vary from matrix to matrix, and application to application. Therefore, it is imperative to automatically
Introduction

select the fastest algorithm and storage scheme for SpMV computation. The majority of the research in the state of the art focuses on improving a single storage scheme or algorithm [117]. Instead, we aim to find the fastest storage scheme and SpMV algorithm for different sparsity patterns by predicting the related performance.

• How to jointly maximize resource utilization and guarantee QoS for different classes of services in dynamic wireless networks with multiple service providers? Future wireless networks should meet heterogeneous service requirements of diverse applications, including interactive multimedia, AI-based applications, augmented reality, and autonomous driving. These applications have heterogeneous resource requirements in terms of bandwidth, latency, and reliability. Unfortunately, RANs face significant challenges to meet the heterogeneous QoS requirements of these emerging applications. For this reason, the concepts of Fog-RANs [135] and network slicing have been proposed [155, 48]: Fog-RANs aim to bring the RAN functionalities and resources close to the end-devices, whereas network slicing provides a flexible way to allocate RAN resources to SPs with different QoS requirements. A major issue with Fog-RAN slicing is related to efficient scheduling and allocation of resources at both the control unit (CU) and the radio unit (RU) to users of multiple SPs [97]. The CU allocates resources to slices whereas the RU allocates the slice resources to UEs such that the SLA between a mobile network operator (MNO) and SPs are satisfied. However, the joint scheduling and allocation of multiple resources in Fog-RANs involving multiple SPs have received limited attention in the literature [167, 11]. Our goal is to address the challenging problem of maximizing overall resource utilization, while also satisfying important economic properties such as fairness, Pareto-optimality, envy-freeness, and sharing incentive.

• How to design an efficient economic mechanism to incentivize service providers in sharing their trained models with others? DNNs are accurate only when trained on a large amount of data. However, suitable input might not always be accessible or may require extensive data collection. Different SPs owning valuable domain-specific datasets can share them with other SPs for training DNN in exchange for a monetary compensation. Unfortunately, sharing data is largely impractical due to privacy concerns. Knowledge sharing overcomes this limitation: the weights from the trained model are shared with interested parties instead of raw data. It is also particularly beneficial when multiple AI-based services are involved in the sharing process. However, these SPs have already put the effort into collecting data and training their DNNs, hence, they might not be willing to participate. Therefore, a suitable incentive mechanism is required; it should
also satisfy key economic properties including Pareto optimality, envy
freeness, sharing incentive, and proportionality. Indeed, the economic
aspects of knowledge sharing in the context of AI-based services have
not been adequately studied. Therefore, we aim to specifically fill
this gap by considering the economics of knowledge sharing in AI
services.

This dissertation proposes novel approaches to answer these research
questions. The publications in this dissertation tackle each of them, as
detailed next.

• Publication I and Publication II devise two distributed offloading
techniques to minimize the overall time for running DNN inference
in dynamic edge/fog networks.

• Publication III presents a tool that leverages deep learning to predict
and execute the fastest SpMV algorithm for a given matrix.

• Publication IV proposes a hierarchical resource scheduling mech-
anism to jointly allocate multiple Fog-RAN resources to slices in
two stages: multi-resource allocation to slices at fog nodes and a
slice-specific resource allocation at each fog node to users.

• Publication V devises an economic mechanism based on Fisher’s mar-
ket [22] for optimal knowledge trading and a weight fusion technique
to incorporate the acquired knowledge.

1.2 Methodology

The research in this dissertation is conducted through a combination of
analytical modeling, mathematical optimization, computer simulations,
and experimental evaluation.

• Economic Modeling. Economic models represent economic pro-
cesses by a set of mathematical equations, variables, and logical as
well quantitative relationships between them. They are employed
to formulate hypotheses about economic behavior that can be tested.
They are then applied to derive implications about economic behav-
ior under the assumption that agents maximize specific objectives
subject to certain constraints, for instance, budget and capacity. We
use matching theory [63] and Fisher’s market [22] in Publication II
and Publication V, respectively.

• Mathematical Programming. An optimization problem consists of
maximizing – equivalently, minimizing – a real function (called objec-
tive function) by systematically choosing input values for variables
Introduction

within a well-defined domain based on a set of constraints. The characteristics of the problem formulation determine the choice of tools to find the optimal solution to the problem. Standard solvers such as MOSEK [119] may be used to obtain an optimal solution when the objective function and all constraints are expressed as linear functions. We use convex optimization – namely, maximizing/minimizing a convex function objective function over convex sets – in Publication V. Furthermore, Publication I, Publication II, and Publication IV formulate optimization problems too. However, due to the higher computational complexity of solving these optimization problems, heuristic algorithms are employed to solve them.

- **Machine Learning.** Machine learning models leverage training data and algorithms to imitate the way humans learn and gradually improve their performance on some sets of tasks. Knowledge is modeled through artificial neural networks, in which neurons are associated certain weights. In supervised learning, weights are adjusted to reduce the error between the model and a known estimate until a desired accuracy is met. In this context, Publication I and Publication III support the execution of DNNs in a network, whereas Publication IV and Publication V employ DNNs and transfer learning (TL) to solve specific problems.

- **Simulation.** We evaluate the proposed models and algorithms in this dissertation through computer simulations in Publication I, Publication II, Publication IV and Publication V. Simulation-based evaluation is employed in most of the state of the art related to the addressed topics due to the large size and scale of the problems being addressed. In most cases, we consider an edge or fog network with a size in the order of hundreds of user devices. Specifically, we chose the simulation parameters that realistically describe urban scenarios. Moreover, we employ custom python simulators built on top of different open-source tools and frameworks to assess the performance of the proposed algorithms: Publication I and Publication II employ the Caffe [77] and PyTorch [132] DNN frameworks, respectively; Publication IV leverages the SliceSim [41] software. In all cases, simulations are carried out according to the independent replication method as many times to ensure adequate statistical significance in the results.

- **Experimental Evaluation.** Experimental evaluation involves an empirical analysis through measurements to evaluate the performance of a system. In Publication III, we discuss sparse matrix-vector multiplications (SpMV) on graphical processing units (GPUs). Specifically, we utilize a node of a supercomputer with NVIDIA K20X GPU to evaluate the performance of different SpMV algorithms. Also
in this case, we carry out several independent replicas of the experiments to ensure adequate statistical significance in the results.

1.3 Research Contributions

The contributions of the five peer-reviewed publications included in this dissertation are summarized next.

Publication I and Publication II address distributed DNN inference acceleration for resource-constrained devices. More specifically, we show how to leverage edge and fog computing to provide fast and scalable DNN inference for embedded IoT devices. Particularly, we minimize the DNN inference time while collaboratively partitioning and distributing computations under dynamic network conditions with performance guarantees. In this respect, we model the DNN inference process for realistic scenarios involving edge computing in Publication I. Moreover, we define optimal inference offloading with load balancing as a multiple assignment problem that maximizes proportional fairness. As the related formulation is hard to solve, we propose an offloading technique – called distributed auction for multiple assignment (DAMA) – that solves the multiple assignment problem in polynomial time and provides near-optimal inference offloading with load balancing in edge networks. Moreover, we extend the model discussed above for fine-grained partition of DNN inference tasks over a fog network in Publication II. We then formulate an optimization problem for optimal assignment of DNN tasks in this context. We also propose two heuristic algorithms: a fine-grained adaptive partitioning scheme that takes into account the network dynamics to divide a source DNN into multiple small pieces; and the Distributed INference Acceleration (DINA) scheme based on swap-matching for offloading DNN inference. DINA characterizes dynamic network conditions as well characteristics of layer types in commonly-used DNNs through an efficient (either sparse or dense) matrix representation that reduces the communication overhead in the network. We evaluate the proposed algorithms through extensive simulations in realistic settings. The obtained results demonstrated that both algorithms achieve a significant reduction in the inference time compared to the state of the art.

Publication III focuses on improving the performance of sparse matrix-vector multiplication (SpMV), which is widely used in several scientific applications such as the sparse DNNs representation discussed in Publication II. Such sparsity varies from matrix to matrix, and from application to application: no single scheme provides the best performance for all sparsity patterns, whereas the performance of an SpMV kernel rather
depends on the structure of the underlying matrices. Hence, automatically selecting the best (namely, fastest) kernel for SpMV computations is imperative. Specifically, we aim at selecting the best SpMV storage scheme and algorithm for execution on GPUs. To this end, we present a tool named DIESEL, which leverages DL to predict and execute the best performing SpMV kernel for a given matrix. The selection is done based on a set of features that have been selected through rigorous analytical and empirical considerations including extensive experiments involving correlation matrix, t-SNE (t-Distributed Stochastic Neighbor Embedding), and 3D cross-feature performance metrics. Specifically, we have trained fully-connected neural networks with DL to predict the best performing storage scheme and kernel for a given input. We evaluate DIESEL extensively on a platform with NVIDIA K20X GPUs. The results obtained from the experiments show that DIESEL provided the best performance among existing tools by considerable margins in terms of prediction accuracy and floating point operations.

Publication IV addresses multi-resource allocation with network slicing in dynamic fog RANs. The RAN domain faces major challenges to fulfill the QoS requirements of emerging applications. Cloud RAN provides a solution but the resulting latency does not always satisfy the QoS requirements of services in 5G. Hence, we leverage network slicing and the Fog-RAN architecture to improve both resource utilization and the profits of SPs delivering services in 5G. We present a comprehensive model of RAN slicing involving uplink/downlink tasks by considering the major features of 5G service types. The task model characterizes realistic streaming scenarios employing file caching, in which users can also decide to stop receiving data before the end of the stream. The joint allocation of multiple types of Fog-RAN resources to multiple slices by an MNO is formulated as a utility maximization problem, where the utility of SPs are characterized in terms of their revenues and costs. To this end, we propose a hierarchical heuristic resource scheduling mechanism named 2L-MRA to jointly allocate multiple Fog-RAN resources to slices in two stages: multi-resource allocation to slices at fog nodes over a given time window and a slice-specific resource allocation at each fog node to users with a much shorter time scale. Moreover, fair and efficient allocation of resources are important to obtain a target quality of service. Consequently, 2L-MRA satisfies important economic properties including fairness, Pareto-optimality, envy-freeness, and sharing incentive. We evaluate the proposed algorithm through extensive simulations based on real-world datasets. The results obtained show that the proposed solution significantly increases the monetary gain of service providers, namely, by 32% to 60% compared to the state of the art.

Publication V addresses knowledge sharing among different SPs offering AI-based services utilizing DNNs. However, SPs might not be willing to
Introduction

share their knowledge if there are no suitable economic incentives due to the effort already put into collecting data and training their DNNs. Hence, we aim at creating a market for SPs to trade and fuse their knowledge to increase their model accuracy. To this end, we propose a novel market-based scheme for knowledge sharing based on DNN weights. In detail, we devise a model based on Fisher’s market to maximize knowledge sharing, defined as the gain in inference accuracy from exchanging DNN weights. Moreover, suitable mechanisms to support the allocation of DNN weights are necessary to ensure participation of the SPs. In this respect, the proposed solution based on the extended Eisenberg-Gale program reaches a market equilibrium and satisfies important economic properties: envy freeness, proportional allocation, and Pareto optimality. The weights acquired by buyers are finally employed to increase their inference accuracy through TL. We prove the optimality and economic properties of the proposed solution and find an optimal solution for the proposed market model through the MOSEK solver. We evaluate it in a distributed intelligence scenario through extensive simulations. The obtained results show that the proposed solution is efficient in clearing the market and that weight fusion with TL significantly increases inference accuracy compared to federated learning, with limited overhead.

1.4 Structure of Dissertation

The rest of the dissertation is structured as follows. Chapter 2 discusses DNN inference acceleration for resource-constrained devices leveraging edge and fog computing. Chapter 3 addresses improving the performance of sparse matrix-vector multiplication (SpMV) by predicting the best storage scheme and SpMV algorithm on GPUs. Chapter 4 targets joint multi-resource allocation in Fog-RANs through network slicing to improve network utilization and the profits of SPs. Chapter 5 discusses a knowledge sharing economy that leverages DNN weights and a market-based mechanism for different SPs to trade and fuse their knowledge and increase their model accuracy. Chapter 6 provides a summary of our contributions and outlines directions for future research. Finally, the original publications are provided at the end of this dissertation.
This chapter presents two techniques to carry out distributed deep neural network (DNN) inference acceleration for resource-constrained devices. More specifically, we show how to leverage edge and fog computing to provide fast and scalable DNN inference for embedded IoT devices. To this end, we devise two offloading techniques to minimize the overall time to run DNN inference. The first technique – called distributed auction for multiple assignment (DAMA) – achieves near-optimal inference offloading with load balancing in edge networks. In contrast, the second technique – named Distributed INference Acceleration (DINA) – provides fine-grained adaptive partitioning and a distributed offloading mechanism suitable for fog networks.

2.1 Background and Motivation

Intelligent applications are becoming more and more ubiquitous due to advances in machine learning (ML) and artificial intelligence (AI), particularly, in deep learning (DL) [105]. These applications have a broad scope that ranges from intelligent assistants (such as Google home and Alexa), and smart city scenarios to video analytics [12, 105]. In most cases, the end devices are resource-constrained embedded systems such as sensors, wearables, smart phones, and smart objects in the Internet of things (IoT) [59]. However, ML algorithms – and DL in particular – require a significant amount of computation to achieve satisfactory results; in fact they are generally executed at powerful servers or data centers [165]. Therefore, several techniques have been proposed to address the limitations of resource-constrained devices [152, 142] when it comes to DNN learning and inference\(^1\). Specifically, these are: (i) reducing precision of operations and operands in DNN [152, 66, 53], (ii) reducing number of operations and model size of DNN [152, 66, 53], (iii) hardware-based optimization and

\(^1\)We only focus on inference since it is more relevant to the considered scenario.
acceleration [30], and (iv) distributed inference at the edge and (or) the cloud [154, 161].

Reducing the precision and the number of operations include: model quantization, which restricts weights into discrete values [33]; reducing the bit-width [71]; and moving from floating point to fixed point [60]. Conversely, reducing number of operations and model size include: pruning, which discards parameters from the source DNN [118, 61]; the use of compact network architectures [70]; and model compression, which removes redundancy [62, 61]. However, these techniques reduce the accuracy of the inference tasks, do not preserve the structure of the model, and may require the availability of special hardware [30].

Computation offloading is a general approach to overcome the constraints of embedded devices through the resources offered by third-party services over the Internet [108, 126]. Such an approach can be leveraged for DNN inference: devices transfer data to one or more powerful servers that then can carry out inference on pre-trained DNNs [68, 179]. Recently, offloading has moved away from the cloud computing paradigm towards the edge of the network [169, 182]. The edge is an infrastructure of server-class devices that are close to end users and can be reached with a low latency [139]. Fog networks push this boundary even further by employing multiple end-devices or near-user edge devices [31]. In addition to offloading, distributing computation to different devices close to the end user reduces the communication overhead while, at the same time, increasing the overall utilization of computing resources in the network.

The major challenge here is that the computation and communication resources provided by edge servers and nearby devices are still limited compared to those in the cloud. As a consequence, they may not be enough to run inference on the original DNN as a whole, especially if the trained model is very large [108]. To overcome this issue the DNN can be divided into multiple partitions, which can be offloaded to multiple servers with an appropriate synchronization of intermediate data. Several approaches – both centralized and distributed – have been proposed in this context [75, 68, 179, 84, 104]. However, they have a number of important limitations. Most schemes built with strong performance guarantees are centralized and make strong assumptions on the system model, such as that DNNs can only be divided into two pieces [68]. Some approaches are distributed and able to handle the complexities of real settings [179], but fail to provide strict performance guarantees in terms of efficient and fair use of resources (see Section 2.2 for more details).

Our research aims to address these limitations. Specifically, we aim to minimize the DNN inference time while collaboratively partitioning and distributing computations under dynamic network conditions with performance guarantees. In this regard, we propose DAMA and DINA to offload inference in edge and fog networks, respectively. Specifically,
we devise a detailed model of the DNN inference process, suitable for realistic scenarios involving edge computing. We also define optimal inference offloading with load balancing as a multiple assignment problem that maximizes proportional fairness. Accordingly, we propose DAMA, a distributed algorithm that solves the multiple assignment problem in polynomial time and achieves strong optimality guarantees. In DINA, we first introduce a fine-grained adaptive partitioning scheme, which takes into account the network dynamics to divide a source DNN in multiple small pieces. Moreover, we present a distributed algorithm – that characterizes dynamic network conditions – based on swap-matching for offloading DNN inference in a fog network.

2.2 Related Work

Several works in the literature have considered DNN inference acceleration through the edge and/or the cloud. The following discussion focuses on those for exact inference, even though schemes tolerating some accuracy loss has been proposed [154, 161, 101, 96].

Different works have proposed centralized solutions for inference acceleration. Among them, Neurosurgeon [84] partitions and offloads DNN computation between mobile devices and the cloud. Specifically, it offloads layers in a linear DNN to the cloud by minimizing both latency and energy consumption, based on real-time monitoring of network traffic. IONN [75] divides a DNN layer-wise into multiple partitions and incrementally offloads them to edge nodes. Its goal is to reduce both transmission and computation time through parallel execution; for this reason it applies the shortest path algorithm on a graph-based characterization of the DNN. Shin et al. [147] extend IONN to derive a fine-grained partitioning scheme based on an efficiency metric, defined as the ratio between the latency reduction and the transmission overhead. DADS [68] leverages a graph-theoretic approach and splits a DNN into two partitions, one offloaded to the cloud and the other to the edge. In particular, DADS determines two types of graph cuts: one to minimize latency for light workloads and another to maximize throughput for heavy workloads. In contrast with these solutions DINA and DAMA are distributed.

Finally, a few works have explicitly targeted distributed schemes for DNN inference. MoDNN [109] considers mobile devices forming a local cluster and connected through high-rate WiFi links. Specifically, MoDNN aims at reducing the synchronization overhead of the devices as they collectively carry out inference tasks. DeepThings [183] devises a partitioning technique that fuses the feature maps in convolutional layers to enable their synchronized execution at multiple edge nodes through a work-stealing algorithm. Similarly, CoEdge [179] applies an adaptive technique that
partitions convolutional layers horizontally and offloads them to edge nodes. Moreover, it carries out cooperative inference by minimizing the energy cost for both computation and communication, subject to deadline constraints. CoopAI [172] employs dynamic programming for partitioning multiple layers, which are grouped into a block and processed in a round. Edge devices work together on the blocks and the intermediate results are assisted by data prefetching. EDGE-LD [171] leverages a MapReduce paradigm to divide a DNN workload among heterogeneous edge devices based on a profiling model that considers both execution time and bandwidth utilization. A layer fusion scheme is also proposed to reduce the communication overhead. Lin et al. [104] consider a three-tier network and DNN partitioned into stages. They propose algorithms to minimize latency and maximize throughput by considering each network tier as a whole. Despite being distributed, all these solutions either leverage heuristics or frameworks that do not provide strong optimality guarantees. In contrast, the theoretical framework developed as part of DAMA achieves near-optimal DNN inference offloading at the edge. Moreover, DINA employs an adaptive partitioning algorithm to split a DNN with sub-layer granularity and a distributed offloading technique based on matching theory, namely, a swap-matching algorithm that targets reducing the total inference time.

Somewhat related solutions have been proposed in the context of virtual network function (VNF) placement and utilization [95], in addition to code offloading [149]. Among them, Jin et al. [80] consider a VNF placement that fosters utilization of computing resources while reducing bandwidth consumption through VNF reuse. Alleg et al. [6] propose VNF placement with flexible resource allocation based on latency, throughput, error rate, and resource utilization. Code offloading [90] is a specific case of the more general task offloading problem, whose objective is to execute a specific code snippet or function on an edge device or cloud server. For instance, MAUI [35] supports fine-grained code offloading at runtime to maximize energy savings based on connectivity constraints of end devices. VNF placement focuses on optimizing the location and routing of network functions by considering network topology, resource availability, and QoS requirements. In contrast, DNN or task offloading in edge networks aims to partially or fully offload computationally-intensive tasks to edge devices or servers to reduce latency, improve QoS, and conserve energy.

### 2.3 Scalable DNN Inference at the Edge

This section first describes the system model for an edge computing scenario wherein resource-constrained end devices offload DNN inference tasks to edge servers. We first characterize the network model, then dis-
Distributed processing of AI models

(a) Distributed processing of AI models
(b) Edge servers
(c) End devices

Figure 2.1. DNN inference offloading at the edge. (a) Reference architecture with heterogeneous end devices and more powerful edge servers. (b) Sample DNN inference tasks \( t_1 \) and \( t_2 \) associated with different devices. Each task relies on a certain DNN architecture with multiple layers \( l_i \). (c) Sample assignment involving DNN tasks and three edge servers.

cuss the task and computational models for the DNN inference process. We then formulate an optimization problem for load-balanced offloading of DNN inference tasks. Finally, we discuss the proposed distributed algorithm that solves the problem in polynomial time.

2.3.1 Network and Task Model

Figure 2.1 illustrates the reference architecture considered. It consists of two components distributed over a certain geographical area: a set of \( n \) edge nodes denoted by \( \mathcal{S} = \{s_1, s_2, \ldots, s_n\} \) and a set of \( m \) devices denoted by \( \mathcal{D} = \{d_1, d_2, \ldots, d_m\} \). We assume that edge servers are connected to each other through dedicated high-speed links [88], hence, the time required to transfer the intermediate results between them are considered to be negligible. In contrast, the end devices communicate with edge servers over a shared wireless channel (i.e., through WiFi) with a total bandwidth of \( B \), according to [68]. In particular, end devices can reach all the edge servers in their communication range.

DNN inference tasks are denoted by a set \( \mathcal{T} = \{t_1, t_2, \ldots, t_T\} \). Specifically, a given task \( t \in \mathcal{T} \) evaluates (i.e., obtains the output of) a DNN. A DNN is represented as a linear graph \( \mathcal{G}^t = (\mathcal{S}^t, \mathcal{L}^t) \), where \( \mathcal{L}^t = \{l^t_1, l^t_2, \ldots, l^t_n\} \) represents the layers of the DNN, i.e., \( l^t_i \) are the layers of the corresponding DNN. Here, \( l^t_1 \) and \( l^t_n \) indicate the input layer and the output layers, respectively; moreover, edges \( (l^t_i, l^t_{i+1}) \in \mathcal{E}^t \) imply that \( l^t_{i+1} \) depends on \( l^t_i \), therefore, it must be evaluated first (Figure 2.1b).

2.3.2 Computation Model

A certain layer can either be executed locally at the end devices or offloaded to edge servers along with the corresponding inputs. An edge server \( k \)
Distributed processing of AI models executes a layer $j$ from user $i$ immediately if it has enough resources available, otherwise it adds it to a local execution queue. The queue follows a FIFO policy and has a maximum size proportional to the computing power of the edge server. The queuing time for layer $j$ from end device $i$ at server $k$ is $T^q_{ijk}$. Layers are not offloaded if the computation time at the edge server, including the time needed for input/output data transfer, is higher than the time $\tau_a$ for local execution at the device.

The computation time of a DNN depends on the type of its layers. We explicitly consider\textsuperscript{2} multi-channel convolution and feed-forward layers [55]. Accordingly, the time $T^e_{ijk}$ for executing a deep learning layer $j$ from device $i$ is $T^e_{ijk} = T^c_{ijk}$ for a convolutional layer and $T^e_{ijk} = T^f_{ijk}$ for a fully-connected layer. The transmission latency required to exchange the inputs, intermediate results, and the outputs between an end device and edge server is

$$T^t_{ijk} = (I^s_{ij} + O^s_{ij})/B$$

(2.1)

where $I^s_{ij}$ and $O^s_{ij}$ are the input and output dimensions. $O^s_{ij} = 0$ if the layer $j+1$ is also executed at server $k$.

Thus the total execution time comprises of the execution, transmission, and queuing time. The total execution time for a layer $j$ offloaded by end device $i$ to an edge server $k$ is:

$$T_{ijk} = T^t_{ijk} + T^e_{ijk} + T^q_{ijk}$$

(2.2)

where: $T^t_{ijk}$ is the time for transmitting the input of layer $j$ to server $k$ and the possible intermediate output from server $k$ back to the device; $T^e_{ijk}$ is the time taken for computing layer $j$ at server $k$; and $T^q_{ijk}$ is the waiting time of layer $j$ at $k$.

### 2.3.3 DNN Inference Offloading with Load Balancing

An end device can offload the execution of individual layers of a given DNN inference task to multiple edge servers. Edge servers can run these layers faster than end devices; however, they may not be able to process requests immediately due to precedence constraints and the load coming from several nodes in the network. This might result in heavy load at few servers while the rest of them are idle. Moreover, running layers at edge servers requires exchanging the input and output of individual layers with end devices over a bandwidth-constrained channel. Therefore, offloading DNN inference must explicitly these factors into account and reduce the total inference time while balancing the load of the edge servers in the network.

\textsuperscript{2}The execution time of activation layer is not considered as part of the total time, since it is negligible.
We model the offloading problem as an assignment problem\(^3\), where the individual DNN tasks requested by the end devices are mapped onto the edge servers (Figure 2.1c). Let $\mathcal{X}$ indicate the assignment of the DNN layers to edge servers. Layer $j$ executed by server $k$ is indicated as the $(j,k)$ pair in $\mathcal{X}$. Not every layer can be assigned to any server and vice versa. Specifically, $\mathcal{L}(k) \subseteq \mathcal{L}$ is the set of layers that can use server $k$, whereas $\mathcal{S}(j) \subseteq \mathcal{S}$ is the set of servers that can run layer $j$. Furthermore, $\mathcal{Y}$ is a feasible assignment as the set of all pairs such that $(j,k) \in \mathcal{Y}$ if and only if $k \in \mathcal{S}(j)$ as well as $j \in \mathcal{L}(k)$.

We now define the problem of offloading DNN inference with load balancing as follows:

\[
\begin{align*}
\text{max}_x & \quad -\sum_{i \in \mathcal{D}} \sum_{(j,k) \in \mathcal{Y}} \log(T_{ijk}) \cdot x_{ijk} \quad (2.3a) \\
\text{s.t.} & \quad \sum_{k \in \mathcal{S}(j)} x_{ijk} \leq 1, \quad \forall j \in \mathcal{L}, i \in \mathcal{D} \quad (2.3b) \\
& \quad T_{ijk} \leq \tau_{ijk}, \quad \forall (j,k) \in \mathcal{X}, i \in \mathcal{D} \quad (2.3c) \\
& \quad n_k^- \leq \sum_{j \in \mathcal{L}(k), i \in \mathcal{D}} x_{ijk} \leq n_k^+, \quad \forall k \in \mathcal{S}, \quad (2.3d) \\
& \quad x_{ijk} \in \{0, 1\}, \quad \forall (j,k) \in \mathcal{Y}, i \in \mathcal{D} \quad (2.3e)
\end{align*}
\]

Equation (2.3a) defines the objective function, which aims to provide load balancing in terms of proportional fairness, a metric widely used for communication systems. This corresponds to maximizing the sum of the logarithms for a certain utility [86], here is inversely proportional to the total inference time $T_{ijk}$ [as in Equation (2.2)]. The total inference time is considered for the layers of all DNN inference tasks offloaded in a feasible assignment $\mathcal{Y}$. The decision variables $x_{ijk} \in \{0, 1\}$ express whether layer $j$ from device $i$ is offloaded to edge server $k$. Equation (2.3b) indicates that individual edge servers may not be assigned any layers. Equation (2.3c) signifies that layer $j$ is only offloaded if its total execution time ($T_{ijk}$) is lower than the local execution time $\tau_{ijk}$ at the device. Equation (2.3d) expresses an upper and a lower bound (i.e., $n_k^+$ and $n_k^-$, respectively) on the number of layers assigned to a given server $k$. Equation (2.3e) states that decision variables are binary. The above problem is a non-linear binary integer programming model, therefore it is NP-hard [21, 36, 122].

\(^3\)Distributed assignment problems have been extensively investigated in other scenarios, including wireless networking and distributed computing [170, 98, 28], but not for DNN offloading.
2.3.4 Distributed Algorithm for DNN Inference

We now propose a distributed algorithm to solve the offloading problem introduced earlier. Publication I details the process of delivering an equivalent formulation of the problem that can be characterized and near optimally solved with the proposed algorithm. Such an algorithm – called distributed auction for multiple assignment (DAMA) – is fully distributed, asynchronous, and operates online. DAMA considers the multiple assignment problem as an economic system wherein an optimal assignment of layers to servers corresponds to reaching an economic equilibrium [19]. It consists of two distinct phases: a forward auction and a reverse auction.

**Forward Auction** The forward auction phase of the algorithm is executed at end device $i$ for each layer $j$ of a DNN inference task. Initially, the device obtains the utility $T_{j,k}$ and the price $p_k$ of each edge server $k$. The price $p_k$ denotes the price to run the server $k$. Initially, the layers and the servers are unmatched, and the device makes a bid for the server offering the highest gain, i.e., highest difference between the layer’s benefit (corresponds to lower execution time) and the server price. The bid is calculated based on the best and the second-best gain a user can achieve by offloading to edge servers. Once a response on the bid – consisting of price and a decision – is received from the server, the device updates the local price with the up-to-date value. If the decision contains a positive acknowledgment (ACK), then the layer is associated with the server. Afterwards, the device listens for bids sent by the edge servers as part of the reverse auction described later. Any new bid arriving from a server is only accepted if the difference between the price and the profit is at least equal to $\epsilon$. If the user accepts a new bid from a server it updates the local profit and sends an ACK with the updated profit. It also sends a negative acknowledgment (NACK) to the concerned server.

**Reverse Auction** The reverse auction phase of DAMA executed at each server $k$. With initial prices and profits of zero, the server listens for bids from devices. When a bid arrives, the server accepts the $n_k^+$ best layers, i.e., those with the highest bids. In such a case, the server updates the profit of each layer and its own price; then, it sends the updated price to the device with an ACK. If the server has already taken $n_k^+$ layers for execution, it still accepts layers but only if the difference between the bid and the price is at least equal to $\epsilon$. If so, the server updates its price, sends it to the device requesting to offload $j$ with an ACK, and to the device of the previously-selected layer $j'_k$ with a NACK. Bids are then sent to devices if all layers are assigned and the server has accepted less than $n_k^+$ layers. First, the server identifies the layer that provides the largest gain, obtains the value for that and for the second-best gain, then derives the bid as done for the user side in the forward auction. It finally places the bid, updates
the local profit with the up-to-date value and possibly the association if the response contains an ACK.

It can be shown that DAMA has a time complexity of $O(mY[\Delta/c])$, where $c$ is the minimum allowed bid increase, $Y$ is the number of feasible assignments, and $\Delta$ denotes the maximum variation in price corresponding to the updates at any end device and server. Moreover, DAMA obtains a solution that is within $mc$ of the optimal assignment.

## 2.3.5 Summary of Results

We employ a custom Python simulator built on top of PyTorch [132] framework to evaluate the performance of DAMA. We consider three different datasets: Stanford Cars [92], Berkeley Deep Drive (BDD100K) [177], and CIFAR-100 [94]. We assume that the DNN tasks are independent from each other and are requested according to a Poisson distribution. We consider three DNN models as benchmarks: NiN, VGG16, and linear AlexNet [148]. We consider a network scenario where static devices are randomly deployed in a square area of size $500 m^2$ according to a uniform distribution. The number of end devices are three times the number of edge servers, as generally considered in the state of the art [44]. We only discuss the results for BDD100K next (see Publication I for a more detailed discussion).

We compare DAMA with four state-of-the-art offloading schemes for DNN inference: Minmax (MM), a modified version of our original optimization problem that minimizes the the maximum execution time; DADS, the partitioning scheme under light workload based on minimum weighted $s-t$ cut of DNN as discussed in [68]; and CoEdge, the distributed horizontal partitioning scheme in [179], which aims to minimize energy cost of offloading subject to latency constraints. The assignments of the DNN tasks are determined whenever a task arrives at the end device and the intermediate data is directly transferred to the relevant device based on this information for all the schemes mentioned above.

Figure 2.2 illustrates the performance of DAMA for the considered benchmarks. Specifically, Figure 2.2a depicts the average inference time (i.e., $T_{ijk}$), as a function of the number of DNN tasks for VGG16 and all the considered schemes. We observe that DAMA obtains the lowest inference time. Moreover, the total inference time significantly increases with the number of DNN tasks for DADS and DINA, as opposed to the remaining schemes. The results clearly demonstrate the scalability of DAMA. In addition, Figure 2.2b illustrates the relative improvement of DAMA in terms of the total inference time with respect to other schemes. We clearly observe that DAMA performs much better than the others, with improvements between 1.14 and 2.23.

Figure 2.3 illustrates the distribution of the DNN layers across the
Figure 2.2. Performance of DAMA: (a) total inference time of the different schemes as a function of the DNN tasks for VGG16, (b) improvement of DAMA in terms of the total inference time for the considered DNN benchmarks.

Figure 2.3. Load distribution for the considered schemes in a network with six edge servers as the number of (a) DNN layers and (b) partitions offloaded to each server.

Table 2.1 further characterizes bad balancing in terms of the Jain’s fairness index [72] for individual edge servers. As observed earlier, DAMA obtains the highest fairness index of about 0.9, therefore, the most fair
Table 2.1. Fairness and convergence time of the different schemes for VGG16.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Fairness</th>
<th>Convergence</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoEdge</td>
<td>0.8523</td>
<td>13.49</td>
</tr>
<tr>
<td>DAMA</td>
<td>0.9120</td>
<td>14.25</td>
</tr>
<tr>
<td>DADS</td>
<td>0.7312</td>
<td>–</td>
</tr>
<tr>
<td>MM</td>
<td>0.7148</td>
<td>16.05</td>
</tr>
<tr>
<td>Consensus-based</td>
<td>–</td>
<td>120.39</td>
</tr>
</tbody>
</table>

allocation. This corroborates our earlier findings related to the distribution of offloaded layers (partitions). Table 2.1 also shows the convergence time as the average number of iterations needed by the different schemes to terminate. We compare CoEdge, DAMA, and MM to a consensus-based algorithm [81]. Since CoEdge is a recursive technique, the number of iterations therein indicate the recursion depth reached. The result shows that DAMA’s convergence is only slightly slower than CoEdge, but still faster than MM.

2.4 Fine-Grained Adaptive Partitioning and Offloading

This section extends the models discussed earlier for fine-grained partition of DNN inference tasks. We then formulate an optimization problem for optimal assignment of DNN tasks in a fog network accordingly. Finally, we discuss adaptive DNN partitioning and distributed inference offloading based on matching theory.

2.4.1 Reference Scenario and Task Model

We consider a network consisting of fog devices (instead of edge servers) in addition to end (i.e., user) devices. In detail, fog devices are denoted by $F \in \{f_1, f_2, \ldots, f_n\}$ and communicate with user devices $u \in U$ in their transmission range, determined by Rayleigh channel fading [64].

A DNN task still consists in evaluating a single DNN, as in Section 2.3. However, each task can be partitioned with sub-layer granularity. Specifically, a given task $d \in D$ can be partitioned into smaller sub-tasks across layers, indicated by $\mathcal{A} = \{a_1, a_2, \ldots, a_A\}$, as illustrated in Figure 2.4. Figure 2.4a provides an overview of the partitioning model, where a sub-task can either be a layer or smaller than a layer. Figure 2.4b illustrates the offloading model for the sub-tasks. Each sub-task can be offloaded to different fog nodes. A segment-based 1D partitioning scheme (Figure 2.4c) is employed to divide tasks with sub-layer granularity for convolutional layers; the partitions $P_1, \ldots, P_n$ of a layer corresponds to individual sub-tasks. A fully-connected layer considered here also includes state-of-the-art
pruned network layers which are sparser than “traditional” fully-connected layers [66]. Hence, partitioning here employs the Compressed Sparse Row (CSR) storage scheme [121]. The CSR storage scheme reduces the amount of bytes stored, hence, decreases the communication latency upon offloading. Moreover, it is applied only when the number of non-zeros in the weight matrix is above a given threshold. Figure 2.4d illustrates a fully-connected layer matrix divided into a partition $P_1$ in dense format and sparse partition $P_2$ in CSR format. Storing the entire matrix in CSR requires only 52 bytes as compared to the 120 bytes required for a dense representation.

### 2.4.2 Communication Model

The Shannon-Hartley theorem [54] is employed to calculate the transmission rate of user nodes as follows:

$$ r_{uf} = \beta \log_2 \left( 1 + \frac{p_u g_{fu}}{\sigma^2 + \sum_{u' \neq u \in U} p_{fu'} g_{fu'}} \right) $$

where: $\beta$ is the total bandwidth assigned to user node $u$; $p_u$ is the transmit power of user node $u$; $g_{fu}$ is the channel gain between the fog node $f$ and
Distributed processing of AI models

user node $u$; $p_{u'}$ and $g_{f'u'}$ are the transmit power and the channel gain of interfering node $u'$, respectively. Consequently, the time taken by a user node to send the DNN partition $a$ to fog node $f$ is:

$$T_{fa}^t = (I_s^a + O_s^a) r_{uf} + \delta$$

(2.5)

where $I_s^a, O_s^a$ are the size of input and output tasks respectively, and $\delta$ is a small edge delay to account for various negligible data transmissions

### 2.4.3 Problem Formulation

We aim to minimize the DNN inference time by partitioning and offloading it to multiple fog nodes, as shown in Figure 2.4b. Specifically, we define the offloading problem as follows:

$$\min_{x_{fa}} \sum_{f \in F} \sum_{a \in A} T_{fa} x_{fa}$$

(2.6a)

subject to:

$$\sum_{f \in F} x_{fa} \leq \theta, \forall u \in \mathcal{U}$$

(2.6b)

$$\frac{p_{u} g_{fu}}{\sigma^2 + \sum_{u' \neq u \in \mathcal{U}} p_{f'u'} g_{f'u'}} \geq x_{fa} \Gamma, \forall u \in \mathcal{U}$$

(2.6c)

$$T_{fa} \leq \tau_a, \forall a \in \mathcal{A}$$

(2.6d)

$$\max_{a \in \mathcal{A}} T_{fa} < \sum_{a \in \mathcal{A}^*} \tau_a, \forall a^* \in \mathcal{P}(\mathcal{A}), \forall u \in \mathcal{U}$$

(2.6e)

$$x_{fa} \in \{0, 1\} \forall u \in \mathcal{U}, f \in \mathcal{F}$$

(2.6f)

where the main objective in Equation (2.6a) is to minimize the inference time $T_{fa}$ when DNN tasks are offloaded to fog nodes. Here, $T_{fa}$ is the total inference time comprising the transmission, execution, and queuing times, as previously discussed in Section 2.3. Moreover, $x_{fa} \in \{0, 1\}$ is a binary variable expressing the offload decision for a task $a$ to fog node $f$ [Equation (2.6f)]. Equation (2.6b) signifies that task $d$ is offloaded to no more than $\theta$ fog nodes at a time. Equation (2.6c) indicates that the rate of transmission should always be greater than a target value $\Gamma$ describing QoS. Equation (2.6d) ensures that the time for task execution at the fog node is always lower than time for executing the same task locally ($\tau_a$). Similarly, Equation (2.6e) states that parallel execution of tasks at fog nodes should take less time than sequential execution of the same tasks locally. In the equation, $\mathcal{P}(\mathcal{A})$ denotes all possible sets of tasks that can be executed in parallel by $\theta$ fog nodes.

The problem above is a non-linear programming problem with a non-convex cost function due to Equation (2.4). Finding an optimal solution is therefore computationally hard [122, 36, 21]. Moreover, the problem assumes that the partition $\mathcal{A}$ is known. Finding optimal solution, however,
requires considering all possible partitions of the main task into sub-tasks, which adds to the complexity of the problem. To address this issue we consider and solve the two separate sub-problems: adaptive partitioning on the one hand and offloading on the other hand. In this regard, we propose two algorithms, DINA partitioning (DINA-P) and DINA offloading (DINA-O) accordingly.

2.4.4 Adaptive Partitioning and Offloading

In the following, we first characterize the dynamic partitioning algorithm DINA-P and later discuss the offloading algorithm DINA-O.

The way layers are partitioned affects the communication overhead, especially for convolutional neural networks that need to process data across partition boundaries. We utilize a segment-based partitioning that requires lower number of redundant values from neighboring partitions for convolutions. Accordingly, we partition convolutional layers into segments.
Distributed processing of AI models along the largest dimension of the input (Figure 2.4c).

When a DNN task $d$ arrives for execution at the user device $u$, DINA-P (Algorithm 1) is executed by the device. DINA-P leverages the concept of utility function to determine the value of partitioning a task. Each fog node $f$ has a service utility for executing a task $a$ of user $u$ as $\varphi_{fa} = x_{fa}(t)(\tau_a - T_{fa}^e - T_{fa} - T_{qa}^f)$, where $\tau_a$ is the delay threshold for the task $a \in A$ and $T_{qa}^f$ is the average queuing latency. The utility is zero when the task is not offloaded, i.e., $x_{fa}(t) = 0$ to a fog node. The fog utility indicates the speed at which a fog node executes a task compared to a threshold time. Similarly, each user $u$ has a utility $\varphi_{uf} = 1/T_{fa}$ associated with each fog node $f$, signifying the delay in executing a user task at a given fog node. The DNN task is dynamically divided into multiple partitions by means of the utility functions. Specifically, the weight matrix associated with the different layers are partitioned into multiple layers $P$, based on the type of layer. A convolutional layer is partitioned as in Figure 2.4c. In case of a fully-connected layer, the matrix is stored in CSR representation if required as discussed earlier, before the partition. Furthermore, the algorithm is separately executed for each parallel path in the DNN model.

Once the partitioning is done, DINA-O (Algorithm 2) is leveraged to offload the partitions to different fog nodes. DINA-O relies on the concept of two-sided exchange stability to find a matching by means of swapping fog-user pairs (see Publication II for more details on matching games). Sorted preference lists are created by all user and fog nodes, based on these utilities calculated before. An initial matching that satisfies the constraints in Equations (2.6b) to (2.6d) is derived through a random assignment of user nodes to fog nodes [143]. If there is an unmatched user after the initial random matching, such user sends a matching proposal to the preferred fog node. The fog node calculates its utility accordingly and accepts the user if it prefers the new matching. Next, a pair of matched user-fog node pair performs a swap involving at least one node if there is a swap-blocking pair and update their utilities. Intuitively, a swap-blocking pair is a pair of user nodes that does not decrease the utility function of any entity involved in the swap. At each iteration, a user node sends a request to its preferred fog node if they are not yet matched to them. Then fog nodes calculate the new utilities for task $a$ from user $u$ and accept the proposal only if their utility is improved by swap matching. If the proposal is rejected, the user nodes makes a request to the next preferred fog node for swapping. The final matching is eventually obtained when no more swaps are possible and offloading occurs accordingly. The convergence and stability of DINA-O are characterized in Publication II.

39
2.4.5 Summary of Results

We utilize a custom network simulator built on top of the Caffe [77] deep learning framework. We consider the BDD100K [177] data set used in Section 2.3 also in this case. We consider four well-known DNN models as benchmarks: NiN and VGG16 as chain topologies as well as AlexNet and ResNet32 as direct acyclic graph topologies. The bandwidth for user transmissions is 10 MHz, similar to that of 5G systems [44]. Computations are characterized by double-precision floating point operations per cycle [42]. All fog nodes have the same computing power whereas the delay thresholds are varied depending upon the size and type of the benchmark DNN. The rest of the simulation parameters are the same as in Section 2.3.

For comparison purposes, three different schemes for partitioning and offloading are considered: random offloading of DNN inference tasks with DINA-P as partition scheme (RANDP); random offloading of the DNN layers without partitioning (RAND); and a greedy algorithm that offloads the DNN layers to the nearest fog node without partitioning (GANC). DINA is also compared against the ECDI-L (DADS) scheme in [68], which was also evaluated in Section 2.3.

Figure 2.5 shows the performance of DINA. Specifically, Figure 2.5a illustrates the total execution time ($T_{fa}$) averaged over all DNN tasks for DINA and the other schemes (RANDP, RAND, GANC) for VGG16. DINA achieves the lowest inference time values. Moreover, its total execution time increases very slowly with the number of tasks, in contrast with the other schemes. This demonstrates the scalability of the proposed offloading scheme, together with the benefits in using DINA-P for partitioning. Note that adaptive partitioning alone is generally beneficial; in fact, RANDP always performs better than other approaches with no partitioning (i.e., RAND and GANC).
Figure 2.5b shows the improvement of DINA in the total execution time as a function of the considered DNN benchmarks. Clearly, DINA outperforms the other solutions, with improvements between 1.7 and 5.2. However, DINA performs worse than GANC for AlexNet and NiN in terms of transmission time. This happens as GANC greedily chooses the nearest fog node, thereby maximizing the rate in Equation (2.4) and reducing interference, as only one user is associated with a fog node in that scheme. Still, this is beneficial only when the parallel paths of the DNN are short (as in AlexNet and NiN), as the related intermediate processing and synchronization are not an issue.

Finally, Figure 2.5c shows the improvement of DINA against ECDI-L [68] in terms of total execution time. The figure highlights how DINA achieves an improvement between 2.6 and 4.2. Such an improvement is higher for the benchmarks other than ResNet32 as these have linear paths; the longer the path, the better the performance of DINA. This happens as ECDI-L does not benefit much from obtaining the maximum cut of graphs with a relatively simple structure.

Figure 2.6 illustrates the load distribution in terms of the amount of computation (in GFLOPS) offloaded to individual fog nodes in the network for the VGG16 benchmark. GANC (Figure 2.6a) obtains the most uneven distribution due to its greedy approach based on physical proximity. As a consequence, fog nodes that are close to many end devices become overloaded; whereas those farther away may not be utilized at all. RANDP results in a more even distribution (Figure 2.6b), due to its random selection policy. Finally, DINA provides the most balanced utilization (Figure 2.6c), with a load almost uniformly spread across fog nodes.

The load balancing can be further characterized in terms of the Jain’s fairness index for individual fog nodes. As observed earlier and further detailed in Publication II, DINA obtains the most fair allocation with 0.9211, 0.9158, and 0.9122 for AlexNet, NiN, and VGG16, respectively; the second-best results are achieved by RANDP with a fairness index...
not higher than 0.6472, whereas GANC performs the worst with values consistently below 0.4.

2.5 Conclusion

This chapter presented algorithms to offload DNN inference in edge and fog networks. Specifically, we first modeled offloading as a multiple assignment problem with load balancing and proposed a distributed auction algorithm (DAMA), that offloads DNN inference to multiple edge nodes. We then extended our model to a distributed algorithm (DINA) based on matching theory, for joint partitioning and offloading of DNN inference tasks in fog networks. Extensive simulations in realistic settings demonstrated that both algorithms achieve a significant reduction in the inference time. A promising future work is represented by evaluating the algorithms in edge or fog testbeds with heterogeneous resource-constrained devices. Other research directions include addressing DNN inference for areas with challenging connectivity due to poor channel conditions and (or) device mobility.
3. Sparse matrix-vector multiplication and linear solvers

This chapter presents our work in improving the performance of sparse matrix-vector multiplication (SpMV), which is widely used in several scientific applications. Our approach is based on AI-based techniques and targets execution on GPUs. Specifically, we aim at selecting the best SpMV storage scheme and algorithm for GPUs. To this end, we present a tool named DIESEL, which leverages DL to predict and execute the best performing SpMV kernel for a given matrix.

3.1 Background

Linear algebra is vital to many fields of science and engineering. Especially sparse linear algebra, considered to be one of seven numerical methods important for science and engineering [10]. It is of prime importance due to its application in important areas such as pruned DNNs [178, 175] (as discussed\(^1\) in Chapter 2), finding the steady-state probability vector of Markov chains [113], communication systems [65], queuing systems [87], and wireless networks [25]. Sparse matrix operations provide the foundation of sparse linear systems. A sparse matrix is any matrix in which the ratio of non-zero elements to the total elements is considerably low. Specifically, matrix \( A \in \mathbb{R}^{m \times n} \), is a sparse matrix when \( z \ll m \times n \), where \( z \) is the number of non-zero elements in \( A \). Sparse matrices that arise from real life problems are large but only include a small number of non-zero elements. Hence, they require special schemes to efficiently store, access, and compute the elements therein.

The coordinate (COO) format [144] (Figure 3.1b) is the simplest sparse storage scheme. It uses three arrays: \( v \), \( r \), and \( c \), each of length corresponding to the number of non-zeroes (\( z \)), to store the matrix non-zero elements.

\(^1\)Chapter 2 introduced DNN offloading techniques to the edge and fog to improve the overall execution performance. This chapter focuses on hardware acceleration and compressed sparse storage schemes to improve the performance of sparse matrix-vector multiplication.
Sparse matrix-vector multiplication and linear solvers

Figure 3.1. Representations of matrix $A$ with: (a) Dense, (b) COO, (c) CSR, and (d) ELL storage schemes.

and their respective row and column indices, respectively. Given an 8-byte floating point number representation and a 4-byte integer representation, the COO format requires $16 \times z$ bytes to store the whole sparse matrix. The compressed sparse row (CSR) format (Figure 3.1c), similar to COO, stores the non-zero values and their respective column indices in arrays $v$ and $c$ (of length $z$), respectively. A third array, $p$, is used to point to the beginning of the each rows. For an $m \times n$ matrix, $p$ of length $m + 1$ stores the offset of the $i^{th}$ row in $p[i]$; $p[m]$ stores $z$. The CSR format requires $12 \times z + 4 \times (m + 1)$ bytes for storage. ELLPACK [56] also known as ITPACK is a sparse matrix storage format well-suited for traditional CPUs as well as GPUs. The sparse matrix $A$ is stored by using different data structures (Figure 3.1d). First, a 2D floating point array, $v$, of size $m \times n^{m_r}$ – where $n^{m_r}$ is the maximum non-zeros per row for a matrix – is leveraged to store the non-zero values; the rows that contain non-zeros less than $n^{m_r}$ are zero-padded. Second, a 2D integer array $c$ of size $m \times n^{m_r}$ is employed to store the respective column indices; again, the rows with $n_r[i] < n^{m_r}$ are zero-padded. The ELL format requires $12 \times m \times n^{m}$ bytes to store a sparse matrix. Clearly, the storage efficiency of ELLPACK reduces for matrices with high variance of non-zero elements per row. The diagonal format (DIA) is only useful for storing sparse matrices with a diagonal structure. It uses two arrays, a $v$ array for storing the values and an offset array $o$ that stores the diagonal offsets of the values from the main diagonal, which has a zero offset. A positive offset indicates super-diagonals and a negative offset indicates sub-diagonals. The size of $o$ is equal to the number of non-zero diagonals of the matrix.

Moreover, sparsity patterns vary from matrix to matrix, and from application to application. Figures 3.2a to 3.2d show the structure of four sparse matrices, each from a different application domain [37]. Figure 3.2e illustrates the performance in terms of Giga Floating Point Operations Per
Figure 3.2. Sparsity structure of four sparse matrices: (a) Freescale1, (b) nd24k, (c) Transport, and (d) tube1; and (e) their performance in GFLOPS for five different kernels.

Second (GFLOPS) of five different SpMV compute kernels\(^2\) (just referred to as kernel later on) for the four matrices; each kernel uses a different sparse storage format: COO, CSR, ELL, HYB, DIA (see Section 3.2 for more details on the individual schemes). It is clear how ELL and HYB provide the highest performance overall. The DIA kernel provides best results for the “Transport” matrix due to the diagonal structure of the matrix, but achieves poor performance for all other matrix structures. The ELL kernel provides better results for all the matrices except for “Freescale1” due to a large difference between the minimum and maximum non-zero elements per row \((n_r)\) in the dataset. The HYB kernel performs best for the “Freescale1” matrix because it is specifically designed for structures with high variance in non-zero elements per row. The CSR kernel provides the best results for the “nd24k” matrix because its non-zero elements are spread across the whole matrix space. We notice that no single scheme provides the best performance for all matrices; the performance of an SpMV kernel rather depends on the structure of the underlying matrices. The other factors that affect performance include architectural features and the storage technique used. Hence, there is a requirement for automatically selecting the best (fastest) kernel for SpMV computations.

In this chapter, we develop a tool called DIESEL (Deep learning-based IterativE Solver tool for large sparS\(E\) Linear equation systems) that predicts and executes the best performing kernel for a given matrix\(^3\). Each kernel utilizes a specific sparse storage scheme and is selected based on a set of features. These features have been carefully selected through rigorous empirical and analytical factors including extensive experiments.

---

\(^2\)The compute kernel is a function compiled for GPUs.

\(^3\)This chapter focuses on kernels for SpMV rather than those for linear solvers. See Publication III for a more detailed discussion on the latter.
involving correlation matrix, t-SNE (t-Distributed Stochastic Neighbor Embedding), and 3D cross-feature performance metrics. We have trained fully-connected neural networks with DL predict the best performing storage scheme and kernel for a given input. We also discuss methods for performance analysis and visualization.

3.2 Related Work

Several works in the literature have considered improving the performance of SpMV computations and iterative solvers. The following discussion focuses on tools utilizing AI to improve performance, even though schemes enhancing the storage formats and the associated SpMV computation algorithms have been proposed as well [13, 102, 168, 5, 46, 180]. We also give a brief overview of fundamental kernels utilized in commercial and open-source solvers. A detailed review of these techniques can be found in [117].

As discussed earlier, the performance of SpMV kernels varies for a given storage scheme based on the matrix sparsity structure. Different works utilize Decision Trees [50]. Sedghati et al. [146] proposed a classification system based on decision trees and employed two feature sets; a basic feature consisting of the matrix characteristics such as density, mean, and standard deviation; and an extended feature set of features that are compute-intensive. Tan et al. [153] propose an auto-tuning SpMV library called SMATER that creates a learning model through data mining and analytical models. They use the C5.0 decision tree algorithm to generate a rule-set pattern with confidence data. A scorecard is used to select the best performing kernel for the predicted storage format. Israt et al. [128] extended [146] by using gradient boosting models (XG-Boost) and ensembles of multi-layer perceptrons (MLP) to predict the best sparse storage format for the SpMV computations of a matrix. The feature sets used are the same as those in [146].

Some works propose techniques based on Support Vector Machines (SVMs). Benatia et al. [14] proposed a method for selecting a storage format to perform SpMV computation on GPUs by using multi-class SVMs. Their proposed features include the number of rows and columns, the total number of non-zero elements, $n_r$, standard deviation, and $n_r$ variance, the density of the sparse matrix, and the difference between the maximum and average $n_r$. They improve this in [15] where they use weighted SVM instead of multi-class SVM. The features used are similar to their previous work but in this case the feature combinations are selected based on the pair of storage formats for creating each model.
3.3 DIESEL: Design and Methodology

The DIESEL tool consists of multiple GPU implementations of SpMV kernels and a Jacobi iterative solver for systems of sparse linear equations. Specifically, the supported SpMV kernels are for COO, CSR, ELL, and HYB. Figure 3.3 overviews DIESEL, which consists of an online and an offline phase. The prediction model is created during the offline phase. Specifically, matrix performance features selected after rigorous empirical and analytical experiments (refer to Publication III for additional details) are extracted from a large dataset comprising 26 real-life applications. The extracted features are stored in a database and are employed for training the prediction model used in the online process. Then, DIESEL takes a matrix and a vector or a sparse linear equation system as input in the online process. It evaluates the model on the set of performance features extracted from the input matrix so as to find the fastest GPU kernel for the SpMV or the iterative solution. The feature database is updated with new features, and the model is retrained periodically following the updates.

DIESEL’s DNN model has six layers, an input layer, four hidden layers, and an output layer. All four hidden layers use ReLU as the activation function. The cost function of the deep network is minimized with the Adam optimizer. We used L2 regularization [16], dropout [17], and early stopping [16] to prevent overfitting. The specific design was chosen after an extensive preliminary evaluation of different network configurations in terms of both number of layers and number of neurons.

Figure 3.3. Overview of DIESEL: features are extracted from a set of training matrices and used to train a model in the offline process; the trained model is used for predicting the storage format for given input and produce the corresponding SpMV results in the online process.
Sparse matrix-vector multiplication and linear solvers

3.3.1 Dataset Formation and Feature Set Selection

The dataset for training and testing of the deep learning prediction model is based on the University of Florida repository of sparse matrices [37]. A subset of 1,056 matrices of different sizes and sparsity structure from 26 application domains are selected (see Publication III for a more detailed discussion on the selection criteria).

We identified fourteen features characterizing SpMV performance (Table 3.1) for DIESEL's DNN after a comprehensive empirical analysis. The selected features enable DIESEL to characterize the structure of a matrix and predict the GPU kernel that provides the best performance for that matrix. The computational complexity of all the features is linear and computation can be carried out in parallel for most of them. Next, we discuss the process and rationale of developing the feature set for DIESEL and empirically analyze it. Specifically, the relationship between the sparsity features and the resulting performance of GPU SpMV kernels are discussed.

Table 3.1. Selected Features From Sparse Matrices

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m )</td>
<td>Number of rows</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>( n )</td>
<td>Number of columns</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>( z )</td>
<td>Total number of non-zero values in the matrix</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>( d )</td>
<td>Density of the matrix, i.e., ( z/(m * n) )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>( \bar{n}_r )</td>
<td>Mean number of non-zero values per row ( (n_r) )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>( \text{sd} (n_r) )</td>
<td>Standard deviation of ( n_r ) in a matrix</td>
<td>( O(2m) )</td>
</tr>
<tr>
<td>( \text{var} (n_r) )</td>
<td>Variance of ( n_r ) in a matrix</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>( z^m )</td>
<td>Maximum of ( n_r ) in a matrix</td>
<td>( O(m) )</td>
</tr>
<tr>
<td>( \overline{z}^m )</td>
<td>Difference between ( z^m ) and ( \overline{z} )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>( d^m )</td>
<td>Mean distance between first and last non-zero values in each row</td>
<td>( O(m) )</td>
</tr>
<tr>
<td>( \bar{e} )</td>
<td>Mean of the number of distinct consecutive ( n_r )</td>
<td>( O(z) )</td>
</tr>
<tr>
<td>( d_m )</td>
<td>Number of matrix diagonals with one or more non-zero values</td>
<td>( O(z) )</td>
</tr>
<tr>
<td>( f_d )</td>
<td>Diagonal fill-in ratio for matrices, i.e., ((d_m \times m)/z)</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>( f )</td>
<td>Fill-in ratio for matrices, i.e., ((m \times z^m)/z)</td>
<td>( O(1) )</td>
</tr>
</tbody>
</table>

We leverage \( t \)-Distributed Stochastic Neighbor Embedding\(^4\) (t-SNE) [159, 107] to reduce the dimensions and visualize the features in our dataset. High dimensional objects are modeled by a 2D or 3D point such that similar and dissimilar objects are represented by nearby and distant points, respectively. Figure 3.4 illustrates the performance profiles in GFLOPS of

\(^4\)The \( t \)-SNE method allows to reduce the dimensions of highly nonlinear spaces and is widely used for visualizing high-dimensional datasets.
Sparse matrix-vector multiplication and linear solvers

the five SpMV kernels for our feature set in the reduced space. The data points that relate to each of the 1,056 matrices are colored according to the kernels that produced the best performance for a given matrix – they indicate the distribution of sparse kernel performance in the reduced space. Specifically, Figure 3.4a shows the performance profiles in a 3D space comprising three features: $\bar{a}^m$, $z$, and $z^m$. The HYB kernel dominates the upper right corner, which represents the highest value for the triplet $\bar{a}^m$, $z$, and $z^m$. This implies that HYB performs better than the other kernels for matrices with higher $\bar{a}^m$ due to a larger $z^m$. HYB also performs better than other kernels when the matrices have high $z$ and medium-high $z^m$. ELL performs better for relatively low values of $\bar{a}^m$ and low to medium values of $z^m$. This correlates with the fact that HYB was introduced due to the inability of ELL to provide satisfactory performance for matrices with high variations in $n_r$. CSR dominates the middle of the plot indicating a better performance for more balanced matrices in terms of the three discussed features. Diagonal matrices have a lower $\bar{a}^m$, hence, we observe some DIA kernels with low $\bar{a}^m$. Similarly, Figure 3.4b depicts the best performing kernels in a 2D space for the $z^m$ and $z$ features. As discussed earlier, HYB dominates regions with higher $z$ and $z^m$. HYB can also be seen towards the rightmost lower corner, since a higher value for $z$ implies larger matrices with a higher value for $\bar{a}^m$. ELL performs best for matrices with high $z$ but with low $z^m$. As seen earlier, CSR is better for medium-sized matrices with average $z^m$ and $z$. The SpMV kernel which dominates a given area in the plot has a higher performance gain compared to the other kernels in the same region. In the areas where a single kernel does not dominate, we find that the performance difference between the kernels is low.

Figure 3.5 illustrates the correlation matrix for the feature set in Table 3.1. Shades of red color signify positive correlation while those of
Figure 3.5. The correlation matrix for the selected features

blue indicate negative correlation; darker colors correspond to extreme values in both directions. Lower correlation is desirable as it implies that the features are relatively distinct and provide a valuable contribution to the feature set in improving the prediction accuracy. The correlation between the majority of features is positive and relatively low (pink shades) whereas only a few negative correlations exist. The features $m$, $n$, $z$, $n_r$, $d_a$, and $d_m$ have a higher correlation – greater than 0.5 (dark red) – due to the linear growth of $z$ with $m$ and $n$. The $d$ feature has a lower negative correlation (shade of light blue) with most other features. Moreover, the positive correlation for $d$ is close to zero. Hence, the density of the sparse matrices is inversely proportional to the number of rows and columns. Our feature set has a low correlation between its individual features. There are some high-correlation features but these have low correlation with the remaining ones. The feature set also shows that our data set is fairly balanced. Finally, the overall low correlation (light colors) of the feature set implies that the member features contribute distinct information for the prediction model.

3.4 Summary of Results

We implement DIESEL by using the CUDA toolkit with C++ along with the related API of Tensorflow [2]. The platform used for the experiments consists of a node with two Intel Xeon E5-2695v2 processors, each with 12
cores running at 2.4 GHz, as well as an NVIDIA K20X GPU. The results reported next are the average from a tenfold cross-validation carried out with the deep learning model. We initially analyze the tool performance in terms of GFLOPS and execution times. The performance of DIESEL is compared with five other kernels; HYB, ELL, CSR, COO, and ELL in terms of these metrics. We later compare the accuracy of DIESEL against five AI-based techniques in the state of the art: the two types of decision trees in [146] (namely, simple cart and BFTree), Multi-class SVM [14], Pairwise weighted SVM [15], and XGBoost [128].

Figure 3.6 illustrates the performance in GFLOPS of the five considered kernels and DIESEL for the entire dataset of 1,056 matrices as a boxplot. The shown results refer to a single SpMV computation involving all matrices. DIESEL provides the highest maximum value (29.25 compared to 20.61 of the second best, CSR), first, second, and third quartiles (12.5, 5.46 and 1.33 compared to 8.79, 4.71 and 0.91 for CSR). DIESEL also obtains a better average compared the other kernels, around 8.5 GFLOPS.
Sparse matrix-vector multiplication and linear solvers

DIESEL also correctly selects the right matrix format (DIA) for a matrix that provides the highest GFLOPS (54.39), compared to 37.11 for the overall second-best kernel, CSR. The average speedup of DIESEL is 3.86, 1.42, 6.07, 1.63, and 1.66 against COO, CSR, DIA, ELL, HYB (respectively).

Figure 3.7 shows the performance in GFLOPS against $\tau$ for each of the 1,056 matrices in our data set for all the five considered kernels and DIESEL. The trend lines in the figure are drawn from the GFLOPS data points using a Generalized Additive Model. The gray area above and below the trend lines shows the corresponding 95% confidence interval. DIESEL (i.e., red data points) provides the highest GFLOPS values among the six kernels for varying non-zero values. Moreover, the trend lines show that DIESEL provides the best performance among all the kernels, followed by HYB, CSR, ELL, COO, and DIA (respectively). Each DIESEL point overlaps with one of the kernels that provides the best performance for the given matrix. As discusses earlier, many matrices cannot to be represented utilizing ELL and DIA, especially at higher non zero values, due to the memory restrictions of the K20 GPUs. For this reason, some data points for ELL and DIA have zero GFLOPS.

Figure 3.8 shows the aggregated execution times of all the considered kernels (COO, CSR, DIA, ELL and HYB) along with DIESEL for all the 1,056 matrices. Specifically, the ELL and DIA kernels only considers 350 and 807 matrices of the 1,056 matrices (respectively) due to the higher memory requirements of these schemes for a certain sparsity. The results presented are for a single SpMV computation involving all matrices. DIESEL attains the lowest execution time (456 ms) compared to all other kernels. The HYB kernel provides the second best performance (744 ms), followed by CSR (1.1 s), ELL (1.2 s), COO (1.3 s), and DIA (1.7 s). The performance of the DIA kernel is the worst, thereby demonstrating its unsuitability for general matrices.

Figure 3.9 shows the performance in GFLOPS of DIESEL and its sub-
Sparse matrix-vector multiplication and linear solvers

Figure 3.9. GFLOPS vs $n_r$ variance and $z$ for (a) DIA; (b) COO; (c) CSR; (d) ELL; (e) HYB; and (f) DIESEL.

kernels (COO, CSR, ELL, HYB, and DIA) against $n_r$ variance and $z$ as a contour plot. The contour lines connect points that have the same GFLOPS, whereas colored contour bands represent ranges of GFLOPS. The GFLOPS vary from near zero to a maximum of 54.39. For better readability, the $n_r$ variance and $z$ have been normalized\(^5\) in the range $[0, 1]$. DIA performs the worst, due to its inability to execute some matrices as well as poor performance for large number of non-diagonal matrices (Figure 3.9a). DIA is followed by COO with majority between 8 and 12 GFLOPS (Figure 3.9b). The CSR kernel (Figure 3.9c) provides higher GFLOPS for low values of $n_r$ variance and $z$; however, it performs poorly for high values of $n_r$ variance and $z$. The ELL kernel (Figure 3.9d) provides satisfactory performance for matrices with high $z$ but performs poorly for high $n_r$ variance due to the large extra-padding required for these. HYB addresses the deficiencies of ELL (Figure 3.9e), although its performance is note adequate for low $z$ and high $n_r$ variance. DIESEL (Figure 3.9f) inherits the performance gain of all the kernels and provides the best results for varying ranges of $z$ and $n_r$ variance, performing better than each individual kernel. In summary, matrices with a diagonal structure achieve better performance with DIA,

\(^5\)The maximum and minimum values of $z$ are 229M and 1,074; for $n_r$ variance they are 12M and 0, respectively.
Sparse matrix-vector multiplication and linear solvers

Table 3.2. Comparison of DIESEL with other AI-based works [146, 14, 15, 128] (based on our dataset and respective feature sets)

<table>
<thead>
<tr>
<th>Technique</th>
<th>Accuracy (%)</th>
<th>ARL (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT (Simple Cart) [146]</td>
<td>69.5</td>
<td>10.8</td>
</tr>
<tr>
<td>DT (BFTree) [146]</td>
<td>67.9</td>
<td>11.5</td>
</tr>
<tr>
<td>Multiclass SVM [14]</td>
<td>80</td>
<td>9.3</td>
</tr>
<tr>
<td>Pairwise Weighted SVM [15]</td>
<td>84.7</td>
<td>7.65</td>
</tr>
<tr>
<td>XGBoost [128]</td>
<td>85.9</td>
<td>8.39</td>
</tr>
<tr>
<td>DIESEL</td>
<td>88.2</td>
<td>4.4</td>
</tr>
</tbody>
</table>

smaller matrices with moderate number of non-zero values performs well with CSR, whereas matrices with a block structure benefit from using ELL and HYB. However, if the maximum non zeros in any row is more than average non zeros HYB performs better than ELL. A more detailed discussion as well as guidelines to decide about the different schemes can be found in [117].

Table 3.2 compares the performance of DIESEL with four state-of-the-art AI-based techniques [146, 14, 15, 128] in terms of prediction accuracy and average relative loss (ARL). Accuracy denotes the fraction of the predictions correctly predicted by the kernel. The ARL – the lower the better – indicates the loss of performance incurred (against ideal performance) due to inaccurate predictions. DIESEL delivers the best accuracy as well as the best ARL; the second-best accuracy is obtained by XGBoost, whereas the second-best ARL is achieved by Pairwise Weighted SVM.

3.5 Conclusion

This chapter presented DIESEL, a tool based on deep learning to predict and execute the best performing SpMV kernel for a given matrix. DIESEL leverages a feature set carefully devised through a rigorous empirical and analytical evaluation. We considered a dataset of 1,056 matrices from 26 different real-life application domains. We proposed a range of new metrics and methods for performance analysis, visualization and comparison of SpMV tools. Extensive experiments on GPUs demonstrated that DIESEL outperforms the state of the art in terms of computational performance, prediction accuracy, and relative loss. A promising future work is extending DIESEL with additional iterative methods and SpMV kernels. Other research directions include considering SpMV support for heterogeneous computing architectures.

This chapter presents a mechanism for joint allocation and scheduling of multiple resources in Fog Radio Access Networks (Fog-RANs). More specifically, we leverage the concept of network slicing and the Fog-RAN architecture to improve network utilization and the profits of service providers delivering generic services in 5G (i.e., eMBB, URLLC, or mMTC). To this end, we propose a hierarchical resource scheduling mechanism named 2L-MRA to jointly allocate multiple Fog-RAN resources to slices in two stages: multi-resource allocation to slices at fog nodes over a given time window and a slice-specific resource allocation at each fog node to users with a much shorter time scale.

4.1 Background

More than half of the Internet traffic consists of Machine-To-Machine (M2M) communication, represented by emerging applications such as video surveillance, remote health-care and smart transportation [74]. These applications have heterogeneous requirements, characterized in terms of the different types of services supported by 5G networks [76]: enhanced mobile broadband (eMBB), ultra-reliable low-latency communications (URLLC), and massive machine-type communications (mMTC). Although both the mobile core and the radio access network (RAN) have been evolving, the RAN in particular faces major challenges to fulfill the QoS requirements of such applications [131].

The cloud radio access network (C-RAN) [134] provides flexible and efficient wireless resource management and allocation to mobile devices. The C-RAN employs a centralized unit (CU) which is connected to radio heads (RUs) through the fronthaul. The CU performs the physical signal processing in a centralized manner, whereas RUs include the radio components and the antennas. However, performing the functions of the physical layer at the CU requires exchanging data and control information with the RUs, thereby increasing latency [131]. For this reason, the fog
Multi-Resource Allocation in Dynamic Fog Radio Access Network Slicing

Radio access network (Fog-RAN) has emerged as a novel architecture to address the shortcomings of the C-RAN [135]. Specifically, the Fog-RAN consists of a fog access point (F-AP) with signal processing, storage and computing capabilities – which are jointly allocated to devices based on their required QoS – attached to each RU. The main idea is to place some RAN functionalities and resources at RUs to provide end-devices with reliable and low-latency Internet connectivity [133].

On a different level, network slicing is a cost-efficient and flexible technique to allocate RAN resources in form of self-contained logical network instances to service providers (SPs) with different QoS requirements [155, 48, 43]. Fog-RAN slicing concerns scheduling and allocation of resources at both the CU and RU to users of multiple SPs [3, 97]. On the one hand, the CU monitors the load of slices in a centralized manner, and upscales or downscales slices to increase resource utilization. On the other hand, each RU allocates the local resources in a slice to UEs, thereby guaranteeing the service level agreement (SLA) between the mobile network operator (MNO) and SPs.

Joint scheduling and allocation of multiple resources in Fog-RANs involving multiple SPs has not been studied adequately in the literature [167, 11]. This chapter specifically addresses this gap by modeling the Fog-RAN slicing problem in such a context and proposing an efficient solution to serve SPs delivering the different types of generic services in 5G (i.e., eMBB, URLLC or mMTC). Both downlink (DL) and uplink (UL) tasks associated with applications belonging to different service types are modeled while considering their needs for caching, computing, and streaming data. A multi-resource allocation to Fog-RAN slices is formulated accordingly as a utility maximization problem. Then, the resource utilization and monetary gain of SPs are determined based on executing the tasks submitted by UEs through Fog-RAN slices. A joint multi-resource allocation and slice admission scheme for dynamic Fog-RAN slicing is finally proposed, with a two-level mechanism (2L-MRA) to assign Fog-RAN resources first to multiple slices and then to UEs admitted to the slices with the goal of maximizing the profit of SPs. In particular, 2L-MRA extends the heterogeneous dominant resource fairness (DRF) [163] level-wise to suit Fog-RANs.

4.2 Related Work

Several works have recently addressed resource allocation in fog computing scenarios [111] and wireless network slicing [3, 47]. Ni et al. [127] propose a dynamic resource allocation scheme by applying priced-timed Petri Nets wherein users select fog resources from pre-allocated resources. Wang and Chen [162] discuss joint optimization of offloading decisions and the allocation of computing resources to minimize latency with a hybrid
genetic algorithm. Liu et al. [106] address joint task scheduling and resource allocation with latency constraints through an alternating convex optimization method. However, these works consider simple settings, such as a single resource pool or only one resource type without slicing. In contrast, this chapter studies multi-resource allocation for Fog-RAN slicing with a focus on UL/DL tasks and the QoS classes in 5G.

Some research has studied resource allocation in wireless network slicing by explicitly targeting user or social utility. Wang et al. [160] aim at maximizing user profit and the social utility of the network to improve resource efficiency. Caballero et al. [26] propose a network slicing game wherein tenants react to the allocation of other tenants by maximizing their own utility. Tran and Le [156] employ a Stackelberg game to model the allocation and pricing of resources for network slicing, with the goal of capturing interactions among access/backhaul SPs and their UEs. Tun et al. [157] address resource allocation in the uplink through a generalized Kelly mechanism and constrained non-linear optimization. Different from all the works above, this chapter targets SPs instead of tenants or access providers, which makes the problem substantially different.

Other research has considered resource allocation in wireless network slicing with a focus on utility of SPs. Fu and Kozat [49] introduce an auction-based framework for resource allocation in RAN slices for uplink transmissions. Kamel et al. [83] consider dynamic allocation of radio resource blocks in LTE network slices by using heuristic algorithms for a downlink scenario. Different from all these schemes, the solution presented here addresses both uplink and downlink scenarios as well as different types of services. Aijaz [4] leverages reinforcement learning and heuristic algorithms to allocate power and resource blocks in 5G networks, for multiple service classes in both uplink and downlink scenarios. Boateng et al. [20] securely maximize the utilities of resource buyers and a seller while balancing QoS satisfaction and resource utilization. Their solution is based on deep Q-networks and blockchain technology. Unfortunately, all the literature discussed above only considers a single type of resource, instead of the more challenging scenario represented by the multiple resources (i.e., bandwidth, storage, and computing) targeted by this chapter.

A few recent works have addressed multi-resource allocation in 5G network slicing [151]. Among them, the most relevant is represented by Fossati et al. [47], which targets the fair allocation of multiple constrained RAN resources to slice instances in critical scenarios. Specifically, they introduce a framework based on the ordered weighted average. However, their work does not consider the economic aspects involved in network slicing. Some recent studies have applied auction techniques for resource pricing and the allocation of multiple RAN resources to slices.
4.3 System Model

Figure 4.1 illustrates the system model of Fog-RAN slicing as considered in this chapter. The system comprises an MNO provisioning the Fog-RAN infrastructure and the related resources such as bandwidth, computing, and storage; a broker, which abstracts network resources and performs network slicing; SPs that acquire network slices with QoS guarantees from the broker to provide DL or UL services to their subscribers; and UEs (e.g., smartphones or IoT devices) that subscribe to SPs to receive certain services.

The network operates in $T$ discrete transmission time intervals (TTIs) over time $t$ ($1 \leq t \leq T$), each with a length of $\Delta t$ (in the order of milliseconds). The network includes a virtual control unit (vCU) pool with radio signaling and processing capabilities in addition to content storage and processing servers. The vCU manages the RAN infrastructure and its resources remotely. The RAN includes multiple RUs, denoted as $\mathcal{R} = \{1, \ldots, r, \ldots, R\}$, which are limited-complexity units (typically realizing only radio-frequency functionalities) and connect to the vCU via fronthaul links. The RUs are connected to the vCU through (fronthaul) fiber-optic links. The average DL and UL link capacities between the vCU and RU $r$ are denoted as $B^d_r$ and $B^u_r$ (bits/s), respectively. The aggregated computing capacity of the vCU is denoted as $P^c$ (CPU-cycles/s). The bandwidth allocated to the DL and UL transmissions in RU $r$ is denoted by $B^d_r$ and $B^u_r$. In addition, each RU $r$ is equipped with one fog access point (F-AP), whose storage and computing capacities are denoted as $C^r$ (bits) and $P^r$ (CPU-cycles/s) [29], respectively.

The broker abstracts the network resources (i.e., bandwidth, storage, and computing) of both the vCU and RUs. Furthermore, it leverages network virtualization functions to manage the life-cycle of network slices including admission, resource allocation, inter-slice isolation, and deallocation. In addition, the broker offers application programming interfaces (APIs) for
Multi-Resource Allocation in Dynamic Fog Radio Access Network Slicing

providers to customize slice allocation to their UEs. Slice hypervisors (located at RU s) periodically (e.g., every 1,000 TTIs) report the status of their resource usage and the active UEs to the vCU, based on which the broker can effectively allocate RU resources to slices.

The system includes a set \( \mathcal{S} = \{ \mathcal{S}_D \} \cup \{ \mathcal{S}_U \} \) of \( S \) SPs. The DL SPs (\( \mathcal{S}_D \)) deliver bandwidth-intensive download services (e.g., eMBB applications), whereas the UL SPs (\( \mathcal{S}_U \)) provide compute-intensive and delay-sensitive upload services (e.g., mMTC applications) to UEs. Therefore, both DL and UL SPs jointly require caching and computing resources (besides bandwidth) in the Fog-RAN to deliver their services.

The system also includes a set \( \mathcal{I} \) of \( I \) UEs (namely, \( \mathcal{I} = \sum_{s=1}^{S} \mathcal{I}_s \)), where \( \mathcal{I}_s \) denotes the subset of UEs subscribed to SP \( s \in \mathcal{S} \) and \( i_s \) indicates a single UE subscribed to SP \( s \). For simplicity, each UE is assumed to subscribe to only one SP (\( \mathcal{I}_s \cap \mathcal{I}_d = \emptyset, \forall s \neq s' \)) and to associate with only a single RU, i.e., the one with the highest channel gain at each TTI, as in [100]. The set of UEs of SP \( s \) associated with RU \( r \) is denoted as \( \mathcal{S}_r^s \). The binary variable \( \psi_{i_s}^r \) indicates the UE/RU association, where \( \psi_{i_s}^r = 1 \) signifies that UE \( i_s \) is associated with RU \( r \) and \( \psi_{i_s}^r = 0 \) if not. The DL rate of UE \( i_s \) in RU \( r \) is:

\[
r_{i_s}^{rd} = a_{i_s}^{rd} B_{rd} z_{i_s}^{rd}
\]

where \( a_{i_s}^{rd} \in [0, 1] \) is the fraction of the DL bandwidth in RU \( r \) allocated to UE \( i_s \) and \( z_{i_s}^{rd} \) is the spectrum efficiency of RU \( r \). Similar to Eq. (4.1), the UL rate of UE \( i_s \) in RU \( r \) is \( r_{i_s}^{ru} = a_{i_s}^{ru} B_{ru} z_{i_s}^{ru} \), where \( a_{i_s}^{ru} \in [0, 1] \) is the fraction of the UL bandwidth in RU \( r \) allocated to UE \( i_s \). Furthermore, the processing capacity of UE \( i_s \) is defined as \( p_{i_s}^r = \phi_{i_s}^r P_r \), where \( \phi_{i_s}^r \in [0, 1] \) denotes the fraction of the computing capability in RU \( r \) allocated to UE \( i_s \). The location and rate of UEs are assumed to be constant during each TTI, even though they may change across different TTIs (e.g., due to UE mobility).

There are \( \hat{S} \) elastic Fog-RAN slices (\( |\hat{S}| = |S| \)) in the system, where a single slice \( \hat{s} \in \hat{S} \) is allocated to each SP (\( s \in \mathcal{S} \)). Each slice \( \hat{s} \) jointly acquires bandwidth, caching, and computing resources in each RU, depending on its service requirements. Following the major service classes defined in 5G networks [47], three types of slices are defined in the network: eMBB (\( \hat{s}_e \)), URLLC (\( \hat{s}_u \)), and mMTC (\( \hat{s}_m \)). Each type of slices has certain latency requirements; for instance, applications may require the end-to-end latency of 1-10 ms or even shorter [1].

The resources allocated to each slice \( \hat{s} \) from RU \( r \) are represented through the vector \( \mathbf{A}_r^{\hat{s}} = (a_{\hat{s}b}^{r}, a_{\hat{s}c}^{r}, a_{\hat{s}p}^{r})^T \), where the tuple denote the bandwidth, caching, and computing resources allocated to slice \( \hat{s} \) in RU \( r \) (respectively). An allocation \( \mathbf{A}_r^{\hat{s}} \) is feasible if RU \( r \) has sufficient resources to satisfy the request. The total resources allocated to slices at a given RU

\(^1\)Consequently, \( s \) and \( \hat{s} \) are used interchangeably in the rest of the chapter.
should not exceed the capacity of RU $r$. Accordingly, the total resources allocated to slice $\hat{s}$ from all RUs should not exceed the cumulative capacity of all $r \in R$.

The minimum amount of resources allocated to UE $i_s$ (once admitted to slice $\hat{s}$) signifies the SLA between the SPs and their subscribed UEs. Therefore, UE $i_s$ is not admitted to slice $\hat{s}$ unless its minimum resource requirements ($i_s$) are guaranteed. The binary variable $\delta^{(t),r}_{is}$ indicates the UE/slice/RU admission, where $\delta^{(t),r}_{is}=1$ implies that UE $i_s$ is admitted to slice $\hat{s}$ in RU $r$ at TTI $t$, and its tasks are served by slice $\hat{s}$ through RU $r$.

### 4.4 Downlink and Uplink Task Model

In this section, we model and discuss the properties of the DL and UL tasks submitted by UEs to SPs.

#### 4.4.1 DL tasks

A DL task includes simultaneous processing and transmission of streaming data (e.g., audio and video files) from the vCU to a UE [74]. The DL SPs publish a common set $F$ of $F$ files, each $f$ of size $l_f$ (bits). The files with the highest request probability are proactively cached at each RU (after fetching from vCU) to locally serve the maximum number of UEs’ requests through F-APs. The binary variable $k^r_f$ indicates file caching (i.e., $k^r_f=1$ if file $f$ is cached at RU $r$), whereas $C^{rd}$ denotes the cache size allocated to the DL services. Accordingly, the sizes of cached files should always satisfy the capacity constraint. Let $\tau^{d,(t)}_{is}=<f, q_f, l'_f>$ describes a DL task submitted by UE $i_s$ ($s \in S_D$) at TTI $t$ to download file $f$, where $q_f$ is the number of CPU-cycles required for processing file $f$ and $l'_f$ is the size of the output file to be downloaded by UE $i_s$.

The completion time of task $\tau^{d,(t)}_{is}$ when the file $f$ is hit and served by RU $r$ corresponds to:

$$t^p_{r}(\tau^{d,(t)}_{is})=\max\left\{\frac{l_f}{p^r_{is}}, \frac{l'_f}{r^rd_{is}}\right\} + \Delta t$$ (4.2)

where $l_f/p^r_{is}$ and $l'_f/r^rd_{is}$ are the total processing and transmission time of task $\tau^{d,(t)}_{is}$, respectively. If file $f$ is not hit in RU $r$, then task $\tau^{d,(t)}_{is}$ is served by the vCU (through RU $r$), and its completion time is:

$$t^v_{r}(\tau^{d,(t)}_{is})=\max\left\{\frac{l_f}{q^v_{is}P^v}, \max\left\{\frac{l'_f}{\alpha^v_{is}B^v}, \frac{l'_f}{r^rd_{is}}\right\}\right\} + \Delta t + \Delta t$$ (4.3)

where $q^v_{is} \in [0,1]$ and $\alpha^v_{is} \in [0,1]$ denote the fraction of the processing and bandwidth resources in the vCU allocated to UE $i_s$ through RU $r$, respectively. The concept of abandon probability is employed to denote the likelihood that a UE interrupts the streaming of a certain file [150]. Such
a probability is leveraged to minimize unnecessary transmission of streaming data that are not relevant to UEs.

4.4.2 UL tasks

UL tasks are characterized based on the requirements of real-world compute-intensive or delay-sensitive UL applications. The UL task \((\tau_{is}^{u(t)})\) submitted by UE \(i_s\) is described in terms of: \(d_{is}\), the size of input data (in bits); \(q_{is}\), the number of CPU-cycles required to complete the task; and \(c_{is}\), the size of the storage space required to cache the output for possible future use. Here, \(q_{is}=d_{is}\eta_{is}\), where \(\eta_{is}\) denotes the processing density (CPU-cycles/bit) of task \(\tau_{is}^{u(t)}\). The execution time of task \(\tau_{is}^{u(t)}\) running locally at UE \(i_s\) is:

\[
t^l(\tau_{is}^{u(t)}) = \Delta t \cdot q_{is}^l
\]

where \(q_{is}^l\) (CPU cycles/s) is the computation capacity of UE \(i_s\). If instead task \(\tau_{is}^{u(t)}\) runs at RU \(r\), its offloading time is:

\[
t^f(\tau_{is}^{u(t)}) = \Delta t \cdot d_{is}^{u(t),u} + \frac{q_{is}^r}{\beta_{isr}^u \cdot P_r}
\]

where the terms on the right side of Equation (4.5) indicate the transmission and execution time of task \(\tau_{is}^{u(t)}\), respectively. Once the execution of task \(\tau_{is}^{u(t)}\) is over, its outcome is cached at RU \(r\).

4.5 Utility Model and Problem Formulation

The utility of SPs is expressed in terms of the difference between the revenue and cost of the DL/UL tasks submitted by their UEs for execution through the Fog-RAN slices. The utility of a task is considered in a single TTI \(t\). The revenue \((Y)\) of task \(\tau_{is}^{d(t)}\) is defined in terms of the reduction in the delay due to the execution through the slice. The utility of SP \(s\) from executing tasks \(\tau_{is}^{d(t)}\) and \(\tau_{is}^{u(t)}\) within TTI \(t\) (respectively) are derived as:

\[
U_{isr}^{d(t)} = Y_r(\tau_{is}^{d(t)}) - \Gamma_r(\tau_{is}^{d(t)})
\]

\[
= \left( (t^f_r(\tau_{is}^{d(t)}) - t^v_r(\tau_{is}^{d(t)})) \Phi_s \right) - \left( (t^f_r(\tau_{is}^{d(t)}) - t^v_r(\tau_{is}^{d(t)})) \Psi_s \right)
\]

\[
U_{isr}^{u(t)} = Y_r(\tau_{is}^{u(t)}) - \Gamma_r(\tau_{is}^{u(t)})
\]

\[
= \left( (t^f_r(\tau_{is}^{u(t)}) - t^l_r(\tau_{is}^{u(t)})) \Phi_s \right) - \left( (t^f_r(\tau_{is}^{u(t)}) - t^l_r(\tau_{is}^{u(t)})) \Psi_s \right)
\]

where \(Y_r(\tau_{is}^{d(t)})\) and \(Y_r(\tau_{is}^{u(t)})\) are the revenue and \(\Gamma_r(\tau_{is}^{d(t)})\) and \(\Gamma_r(\tau_{is}^{u(t)})\) the cost of SP \(s\) from executing tasks \(\tau_{is}^{d(t)}\) and \(\tau_{is}^{u(t)}\) within slice \(s\) through RU \(r\) respectively. \(\Phi_s\) and \(\Psi_s\) is the price in PCT units – a PCT is the price in $ for downloading 1 MB of extra data during \(\Delta t\).
The joint allocation of multiple Fog-RAN resources to slices is formulated next as a utility maximization problem.

**Problem 1** (Multi-Resource Allocation to Fog-RAN Slices). The Multi-Resource Allocation to Fog-RAN Slices (MRAFS) problem is defined as:

\[
\max_{\psi, \delta, \alpha, \phi, k} \sum_{s \in \mathcal{S}} \sum_{r \in \mathcal{R}} \sum_{i \in \mathcal{I}} U_{isr}^{d, (t)} + \sum_{s \in \mathcal{S}} \sum_{r \in \mathcal{R}} \sum_{i \in \mathcal{I}} U_{isr}^{u, (t)} \tag{4.8a}
\]

subject to:

\[
\sum_{r \in \mathcal{R}} \psi_{is} \leq 1, \forall i \in \mathcal{I}, s \in \mathcal{S} \tag{4.8b}
\]

\[
\sum_{r \in \mathcal{R}} \delta_{is}^{(t), r} \leq 1, \forall i \in \mathcal{I}, s \in \mathcal{S} \tag{4.8c}
\]

\[
\sum_{s \in \mathcal{S}} \sum_{i \in \mathcal{I}} \psi_{is} \delta_{is}^{(t), r} a_{is}^{rx} \leq 1, \forall x \in \{u, d\}, r \in \mathcal{R} \tag{4.8d}
\]

\[
\sum_{s \in \mathcal{S}} \sum_{i \in \mathcal{I}} \psi_{is}^{r} \delta_{is}^{(t), r} \phi_{is} \leq 1, \forall r \in \mathcal{R} \tag{4.8e}
\]

\[
\sum_{f=1}^{F} l_{f} \leq C^{rd}, \forall r \in \mathcal{R} \tag{4.8f}
\]

\[
\sum_{r \in \mathcal{R}} \psi_{is}^{r} \delta_{is}^{(t), r} r_{is}^{m} \geq c_{is}^{m}, \forall x \in \{u, d\}, i \in \mathcal{I}, s \in \mathcal{S} \tag{4.8g}
\]

\[
\sum_{s \in \mathcal{S}} \sum_{i \in \mathcal{I}} \sum_{k \in \lambda_{is}} \psi_{is}^{r} \delta_{is}^{(t), r} p_{is}^{l} \geq p_{is}^{m}, \forall r \in \mathcal{R} \tag{4.8h}
\]

\[
\psi_{is}, \delta_{is}^{(t), r}, h_{f} \in [0, 1], \forall i \in \mathcal{I}, s \in \mathcal{S}, f \in \mathcal{F} \tag{4.8j}
\]

\[
\alpha_{is}^{rd}, \alpha_{is}^{ru}, \phi_{is}, \delta_{is}^{p} \in [0, 1], \forall i \in \mathcal{I}, s \in \mathcal{S} \tag{4.8k}
\]

The objective of the problem is an efficient and fair allocation of network resources to the maximum number of UEs in the slices, thereby increasing the profit of SPs in Equation (4.8e). In particular, the utility maximization problem relies on the binary variables \(\psi\) and \(\delta\) to denote the UE/slice/RU association. Furthermore, the formulation includes the fractional resource allocation variables \(\alpha\), \(\phi\), and \(k\). Specifically, Equation (4.8b) signifies the association between UEs and RUs, while Equation (4.8c) the one between UEs, slices, and RUs. Equation (4.8d) states that the fractional DL and UL bandwidth capacities at each RU (respectively) should be less than one. Similarly, Equation (4.8e) restricts the fractional processing capacity at each RU to values below one. Equation (4.8f) states that the caching
capacity at each RU should not be exceeded. Equation (4.8g) indicates the minimum DL and UL bandwidth requirements for each UE. Equations (4.8h) to (4.8i) indicate the caching and processing requirements for each UE. Equation (4.8j) specifies the binary variables and Equation (4.8k) the fractional variables. Problem 1 is an instance of a multi-dimensional knapsack problem, which is NP-hard [85].

4.6 Two-Level Multi-resource Slicing

In this section, we propose a two-level multi-resource allocation (2L-MRA) mechanism for dynamic Fog-RAN slicing. 2L-MRA leverages the dominant resource fairness (DRF) framework [52] for resource allocation, a generalization of the max-min fairness rule to heterogeneous resources [18]. Accordingly, the dominant resource for each UE is defined as the most heavily required resource for executing a task or availing a service. Furthermore, the dominant share for each UE is defined as the fraction of the dominant resources allocated to a UE. DRF aims to find a maximum allocation that equalizes the dominant share of each UE.

A two-level scheduling approach is employed by 2L-MRA to allocate the network resources to slices and then to the UEs in each slice. Such an approach leverages time windows composed of multiple TTIs (Figure 4.2). Scheduling takes place at two levels: at the vCU and at the RU. In the CU-level scheduling, the broker dynamically allocates the resources of RUs to active slices at the beginning of every time window $w$ with duration $W$, such that $0 \leq W \ll T$ and $\Delta t \ll W$. Next, in the RU-level scheduling, the slice hypervisors (at each RU) allocates the resources of each slice to its active UEs at the beginning of every TTI $t$ within each time window $w$. The time windows have a significantly longer time scale, which is at least one order of magnitude higher than the TTI duration. The following overviews the scheduling process at both CUs and RUs. Additional details on how to determine the related resource allocation and on the economic properties.
of 2L-MRA can be found in Publication IV.

**CU-level scheduling** At the very beginning \((w=0)\), the broker initializes the demand vector for each RU by using the minimum value of supported UEs based on the slice classes. At the beginning of each time window \(w\) (for \(w>0\)), the broker estimates the resource demand (bandwidth, compute, and storage) for slices in each RU for time window \(w\). The broker then updates the demand vector \(D^s_\hat{s}\) by employing the time average demand rate for a slice at RU. The actual demand in the previous window is obtained based on usage statistics and the demand of the resources for each slice at each RU is sent to the broker at every TTI. Next, the broker allocates the resources by considering the DRF, through which it determines the fractional resources \(a^s_\hat{s} \in A^s_\hat{s}\) to be assigned to each slice \(s\) at RU \(r\). Specifically, the broker jointly allocates resources across all the slices at RU \(r\) to achieve max-min fair allocation for the dominant resources \(\tau^s_\hat{s}\). Accordingly, the non-wasteful resource allocation for slice \(s\) at RU \(r\) is \(A^s_\hat{s}=(a^s_\hat{s}, \theta^s_\hat{s}, \varphi^s_\hat{s}) = o^s_\hat{s} \cdot d^s_\hat{s}^n\), where \(o^s_\hat{s}\) is the dominant share slice \(\hat{s}\) receives at RU \(r\) under \(A^s_\hat{s}\) and \(d^s_\hat{s}^n\) the normalized demand (with respect to dominant resource). The broker allocates the maximum dominant share \(o^s_\hat{s}\) to each slice under the fairness and capacity constraints. Progressive filling\(^2\) is employed to find \(o^s_\hat{s}\).

**RU-level scheduling** Once the broker determines the allocation matrix \(A^s_\hat{s}\) for each slice \(\hat{s}\) at RU \(r\) (at the beginning of each window), slice hypervisors receive the resource demands \(d^s_{i\hat{s}m}\) (namely, the minimum resources required for slice admission) from each UE \(i\) at the beginning of every time slot within TTI \(t\). The slice hypervisors at each RU \(r \in R\) initialize the UE resource allocation matrices \(A^{i\hat{s}} \in \mathbb{R}^{[|S^i|] \times 3}\) with zeros, where row \(a^{i\hat{s}}_{i\hat{s}}\) determines the fractional resources assigned to UE \(i\) at RU \(r\) for the slice \(\hat{s}\). Similar to CU-level scheduling, each slice hypervisor at \(r \in R\) jointly allocates the resources to all the UEs admitted to the slice while achieving max-min fair allocation for the dominant resource. Accordingly, the non-wasteful resource allocation for UE \(i\) at slice \(\hat{s}\) is \(a^{i\hat{s}}_{i\hat{s}}=(a^{i\hat{s}}_{i\hat{s}}, \theta^{i\hat{s}}_{i\hat{s}}, \varphi^{i\hat{s}}_{i\hat{s}}) = v^{i\hat{s}}_{i\hat{s}} \cdot d^{i\hat{s}}^n\), where \(v^{i\hat{s}}_{i\hat{s}}\) is the dominant share UE \(i\) receives at slice \(\hat{s}\) at RU \(r\) under \(a^{i\hat{s}}_{i\hat{s}}\). Furthermore, \(d^{i\hat{s}}^n\) is the normalized demand. However, obtaining \(v^{i\hat{s}}_{i\hat{s}}\) is not feasible when there are not enough resources to satisfy the minimum requirements of UEs/SPs. Hence, in addition to progressive filling [52], an admission queue is maintained: the maximum number of UEs/SPs (respecting SLAs) is considered for admission to slice \(\hat{s}_q\) at RU \(r\) on a first-in-first-out basis and later obtains the resources by finding \(v^{i\hat{s}}_{i\hat{s}}\).

When the utility of the SPs for the considered UL/DL tasks in the slice admission list is less than zero (no profit), then the tasks are served through the backhaul or by other means at the given TTI. These tasks are also removed from the potential slice admission list and replaced with

\(^2\)The progressive filling algorithm achieves max-min fairness in a system where resources can be allocated in arbitrarily small amounts [18].
waiting tasks from the queue. The tasks in queue with higher utilities are swapped with tasks in the potential list with lower utilities (including resource allocation). Consequently, the users in the potential list are admitted to the slices. The remaining services in the queue at the given TTI are served through the backhaul or by other means. This ensures that the slices $\hat{S}$ only admit the UEs which can satisfy the minimum required resources. Once $a_{is}^r$ is determined for each slice $s$, then each UE $i_s$ is provided additional fractional $(\alpha_{rs}^r - \alpha_{is}^r)|I_s|$ bandwidth, $(\theta_{rs}^r - \theta_{is}^r)|I_s|$ caching, and $(\varphi_{rs}^r - \varphi_{is}^r)|I_s|$ processing resources. This is because the initial resource allocation is non-wasteful, hence, the UEs at most receive only the demanded resources for a service. The excess resources are fairly allocated to the UEs of that slice to improve the QoS.

2L-MRA allocates resources in polynomial time. It can be shown that 2L-MRA satisfies the following economic properties: envy freeness, when a UE (and a slice) does not prefer the allocation vector of another UE (and slice) to its allocation; Pareto optimality, where it is not possible to increase the resource of a UE (and a slice) without decreasing the resource for other UEs (and slices); slice sharing incentive, where slices should be able to serve more UEs compared to a uniform allocation where each slice is assigned the same amount of RU resources.

4.7 Summary of Results

We employ a custom network simulator built on top of the SliceSim software [41], a python-based framework for network slicing in 5G. The scenario illustrated in Figure 5.1 is considered, with $|S|=6$ SPs, equally divided between eMBB (DL), URLLC (UL), and mMTC (UL). Simulations last for $T=1 \times 10^7$ TTIs, where each $\Delta t=1$ ms. Additional details about the simulation setup are reported in Publication IV.

We employ two datasets to characterize the service utilization of the DL SPs: the Live Streaming Sessions Dataset [137] and the Music Streaming Sessions Dataset [24]. The services provided by the DL SPs are randomly selected from the datasets according to a uniform distribution. Furthermore, the abandon probability is calculated for each UE subscribed to the service from the associated data. The UL tasks are assumed to be independent of each other; they arrive at users according to a Poisson distribution at the rate of 10, 20, and 30 tasks/s for the eMBB, URLLC, mMTC services (respectively). The minimum resource requirements for the eMBB, URLLC, and mMTC slices (for both UL and DL) are randomly selected from a subset of 20 real-world Amazon instances [7] suitable for the three slice categories (e.g., c4.8xlarge is compute-optimized, r4.16xlarge is memory-optimized, i3.8xlarge is storage-optimized). The resource capacity of each F-AP is randomly selected from a set of M4 and M5 Amazon EC2
Multi-Resource Allocation in Dynamic Fog Radio Access Network Slicing

We compare 2L-MRA with the following schemes: static slicing with an equal allocation (SSE) [26], which allocates the resources to the slices once and remains the same over time; static slicing with a proportional allocation (SSP) [125], which allocates resources proportionally to the slices based on their demands (i.e., the number of UEs) and also remain the same over time; dynamic slicing with dynamic hierarchical resource allocation (DSDHR) [100], which allocates resources dynamically to slices along with user admission at each window \( w \) and UEs are dynamically allocated resources every TTI for the considered slice classes; dynamic slicing with proportional allocation (DSP), a variant of 2L-MRA in which resources are dynamically scheduled and allocated to the slices based on their demands (i.e., the number of UEs) at each window \( w \). All the schemes above guarantee the minimum resource requirements to UEs through the slices, similar to 2L-MRA.

Figure 4.3 illustrates the total utility achieved as a function of different parameters. In particular, Figure 4.3a shows the impact of slice scheduling on the total utility over time, expressed as windows during a representative simulation run, for the different schemes. Both SSE and SSP have a slower rate of utility change. In particular, the rate of utility change for SSE, SSP, DSDHR, DSP, and 2L-MRA is 4.17, 6.77, 10.9, 11.41, and 18.23 per window (respectively). Such a higher rate change for 2L-MRA, DSP, and DSDHR is due to the dynamic slice scheduling at each window. DSP performs better than DSDHR as the latter is unable to efficiently assign resources to slices.

Figure 4.3b illustrates the total utility as a function of the number of SPs for the considered schemes. As the number of SPs increases, the utility also increases in all cases. The total utility with 2L-MRA is high, due to the dynamic scaling of the slices and the more effective allocation of resources to UEs across slots per TTI. The two static scheduling schemes, SSE and SSP, have the lowest utilities. Instead, 2L-MRA provides the highest utility in all cases, with a gain of 63%, 53%, 42%, and 33% compared to SSE, SSP,
DSDHR, and DSP (respectively) when there are 16 SPs.

Figure 4.3c illustrates the total utility as a function of the task arrival rate ($\lambda$) for the considered\(^3\) schemes. Such utility increases as the number of tasks increases as well. Moreover, 2L-MRA obtains the highest utility, with an increase of 50.67% over the second-best scheme, DSP; both DSDHR and SSP perform very poorly. The rate of utility increase, instead, is inversely proportional to the task arrival rate. This is due to the reduced availability of resources for newly arrived tasks, which are then served through the backhaul. The average increase for each SP in 2L-MRA until $\lambda = 100$ is $11.44$ per task while the average increment per task for each SP is $1.4$ between $\lambda = 100$ to $\lambda = 1,000$. In contrast, DSP achieves an average utility of $7.53$ per task and $0.927$ per task for a given SP.

Figure 4.4a depicts the optimality gap of 2L-MRA for a small network topology with $|\mathcal{Z}| = 6$ and $|\mathcal{S}| = 3$. The figure shows the total utility of both 2L-MRA and the optimal solution of Problem 1 obtained through the IBM CPLEX solver [34] with the branch-and-cut algorithm. The utility of 2L-MRA is close to the optimal solution, exhibiting a gap between 1.1% and 3.5%, with an average of 2.2%. It is worth recalling that 2L-MRA has a polynomial time complexity, whereas solving Problem 1 becomes practically infeasible for large networks.

Figure 4.4b shows the Jain’s fairness index [73] of the different schemes with respect to the utility obtained by the SPs as a box plot. The obtained results clearly show how 2L-MRA outperforms the other schemes, with a median index of 0.90, and most of values ranging between 0.94 and 0.88. DSP has a median fairness index of 0.84, below the 25th percentile of 2L-MRA, with a similar spread. The other schemes – DSDHR, SSP, and SSE – obtain significantly lower values of fairness (of approximately 0.7) and exhibit much higher deviation.

Finally, Figure 4.4c shows the average resource utilization ratio with respect to the individual resources across all the slices at RUs. 2L-MRA\(^3\)SSE is not reported for better readability, as the related utility is very low.
achieves the highest resource utilization with 96%, 98%, and 99% of bandwidth, storage, and vCPU usage – as opposed to the 90%, 86%, and 50% utilization of the same in the second-best scheme, namely DSP. It is clear that, except for 2L-MRA, a higher utilization of one or two resource types results in a poor utilization of the remaining resources. This correlates with the achieved utility, as previously discussed; consequently, 2L-MRA is particularly effective while considering multiple resources for network slicing.

4.8 Conclusion

This chapter proposed 2L-MRA, a utility-based technique for joint allocation of heterogeneous resources in Fog-RAN slicing. 2L-MRA considers both UL and DL SPs as well as different classes of services to maximize utility over time. It has a polynomial time complexity and achieves important economic properties: Pareto optimality, envy freeness, and sharing incentive. Extensive simulations in realistic scenarios have demonstrated that 2L-MRA significantly increases utility in varying network conditions due to dynamic resource allocation to slices and fine-grained resource scheduling for UEs, resulting in higher utilization and better QoS than the state of the art. Future research could consider how the pricing of different resource types affects the economics of multi-resource allocation with multiple SPs. Another promising direction is given by the design of techniques based on machine learning to allocate heterogeneous resources according to different application requirements.
This chapter presents a knowledge sharing scheme that leverages DNN weights and a market-based mechanism for scenarios involving different service providers (SPs). More specifically, we aim at creating a market for SPs to trade and fuse their knowledge to increase their model accuracy. To this end, we take an economic approach and devise a model based on Fisher's market [22] for optimal trading and a weight fusion technique to fuse the acquired knowledge.

5.1 Background

Services leveraging Artificial Intelligence (AI) – ML in particular – are having an increasing impact in our daily life [105]. Indeed, most AI-based services employ deep neural network (DNN) to realize intelligent functions. Among them object detection and speech processing are widely used for use cases ranging from smart homes to autonomous driving [115].

Attaining adequate accuracy for DNN models require training on large-scale public or private data sets and requires extensive data collection [184]. However, most of the publicly-available datasets may not be representative of data in real applications [114]. Data sharing is an alternative, in which the owners of valuable domain-specific datasets share them with other parties in exchange for a monetary compensation [141]. However, data sharing leads to privacy concerns related to origin of the data [89] and it involves the problematic process of finding an agreement between different parties [27]. Knowledge sharing is an alternative approach wherein the knowledge built by using the data – represented by the weights of a pre-trained DNN– is shared rather than the actual data [184]. In this context, transfer learning is an effective technique to employ the weights of a given DNN with a different one [130]. Accordingly, a pre-trained model for a given task is used as a starting point or as a feature extractor for a target DNN. A subset of the old layers is frozen during training and the remaining layers are then re-trained on a different task with a new training
A different approach is federated learning, in which different devices train a model based on local data and then share such a model with others through a central coordinator to gain global knowledge [112].

Knowledge sharing is particularly beneficial when multiple AI-based service providers (SPs) are involved. In fact, the knowledge from DNNs trained for a specific use case could as well be employed in other applications. For instance, an intelligent transportation system may run different types of DNNs for object recognition, which might also be applied to smart manufacturing. However, SPs might not be willing to participate if there are no suitable economic incentives [38]. This is mainly because of the effort already put in collecting data and training their own DNNs.

Indeed, the economic aspects of knowledge sharing in the context of AI-based services is very important, however it has not been adequately studied in the state of the art. In this chapter, we specifically address this gap by considering the economics of knowledge sharing in AI services through a market-based approach. We propose a novel market-based scheme for knowledge sharing based on DNN weights. It considers multiple service providers trading weights of DNNs pre-trained to specific use cases. In particular, a weight market mechanism is devised to trade and fuse knowledge of different SPs. In particular, it devises a model based on Fisher’s market [22] to maximize knowledge sharing. The proposed solution satisfies several important economic properties including envy freeness and proportional allocation. The weights acquired by buyers are finally employed to increase their inference accuracy by means of transfer learning.

5.2 Related Work

Economic models for resource allocation have been largely studied in the literature. More specifically, several works have considered competition among SPs in different scenarios. Huang et al. [69] propose a resource allocation model based on game theory that considers the competition between cloud and network SPs. Feng et al. [45] study monopoly, duopoly, and oligopoly to find optimal pricing for cloud services. Similarly, other works focus on determining the optimal prices and resource allocation in cloud, edge and fog scenarios. Prasad et al. [138] propose an online Fisher’s market that dynamically determines the price and the integral allocation of diverse resource types based on the supply and demand in a certain period of time. Li et al. [103] discuss a price-incentive reverse auction mechanism wherein cloud resources are sold to users by trading off the interests of both entities. Peng et al. [136] introduce a reverse auction mechanism for resource allocation in vehicular fog networks. Li et al. [101] address allocation of crowdsourced edge resources to competing
users through a truthful double auction mechanism for dynamic resource pooling at the edge. Wang et al. [164] leverage a stochastic learning scheme to achieve a Nash equilibrium in a distributed manner in the context of edge intelligence by minimizing the average age of information. In contrast, we devise a market model of knowledge sharing for the specific scenario in which different SPs provide AI-based services. Rasouli and Jordan [141] characterize markets in which agents exchange data to carry out machine learning tasks. The authors consider trading data for money and characterize such a process as a network formation game. Instead, we leverage a Fisher’s market formulation for optimal knowledge sharing rather than deriving optimal prices.

In addition to resource allocation, different learning schemes have been addressed in the literature. Among them, transfer learning (TL) is based on leveraging knowledge acquired for one (base) task to solve a related task in a different (target) domain [130]. Several works propose new TL architectures [176, 166, 181]. In contrast, we do not develop a new TL architecture, but rather apply TL to combine the weights of pre-trained DNNs obtained as part of a knowledge sharing process. Several works leverage federated learning (FL) to solve privacy issues in conventional centralized solutions [112]. In FL, devices train a local model then sends its parameters to a central coordinator that aggregates and re-distributes the updated model back to the devices. Accordingly, recent works have focused on learning and improving a single global model driven by a central orchestrator [181, 67]. Despite its privacy preserving nature, FL generally has a substantial communication overhead, since the server sends the model to all clients repeatedly [9, 145]. Instead, our work relies on exchanging and combining the weights of pre-trained DNNs along with transfer learning. Consequently, it significantly reduces the communication overhead while improving the accuracy of local predictions at the same time. This work is one of the first to address economic aspects of knowledge sharing in AI-based applications offered by multiple SPs.

In the different domain of participatory sensing [51], several works have discussed diverse schemes to provide incentives to users. RADP [99] is a “winner take all” scheme leveraging a reverse auction-based incentive mechanism: participants sent their incentive expectations to the platform and those with the lowest expectations are chosen as auction winners to carry on the sensing task. Koutsopoulos et al. [91] use a Bayesian model to guide participants in realizing their most profitable bidding prices and participating levels by submitting a different amount of data samples or diverse data types. Yang et al. [173] propose a model in which the platform is the leader and decides on the payment to each participant, while participants could only tailor their actions to the platform – the incentive negotiation procedure is modeled as a Stackelberg game. However, most of these schemes only determine an incentive for participation in sensing.
Figure 5.1. (a) Reference scenario: multiple providers offer AI-based services, relying on different DNN architectures. (b) Service providers sell and buy the weights associated with the DNNs they use as a form of market-based knowledge sharing based on transfer learning. (c) Acquired weights are used by buyers to update (part of) the knowledge in their DNNs so as to increase their inference accuracy.

Instead, our work not only incentivizes both buyers and sellers, but also determines a fair sale and allocation of weights for sharing knowledge.

5.3 Weight Market for Knowledge Sharing

This section first describes the reference scenario and the main features related to the considered market model. It then formulates an optimization problem for optimal knowledge sharing in such a context.

5.3.1 Reference Scenario

Figure 5.1 illustrates the reference scenario under consideration. It consists of a network with multiple service providers (SPs) offering AI-based services to end users. These services target diverse applications, requiring different types of inference (i.e., object detection, text extraction, time series forecasting, image segmentation). A DNN is trained for each of these specific use cases. However, the knowledge of a certain DNN could also help other services relying on the same functions. For instance, the existing knowledge on image segmentation for assisted driving could be employed for video surveillance too (Figure 5.1b). Specifically, we leverage transfer learning [130] here to address knowledge sharing wherein multiple SPs offer the weights of their DNNs for sale to others. The SPs are denoted by the set \( \mathcal{S} \) and divided into buyers \( s \in \mathcal{S}_B \) and sellers \( s \in \mathcal{S}_S \), such that \( \mathcal{S} = \mathcal{S}_B \cup \mathcal{S}_S \). Each \( \mathcal{S}_S \) offers the weights of \( R \) different DNN architectures for sale, where \( r \in \mathcal{R} = \{1, 2, \ldots, R\} \) is the actual type of DNN, for
Knowledge sharing economy

instance, AlexNet, ResNet32, or VGG16 [116]. An orchestrator coordinates the knowledge sharing process while operating as an inference serving system [78]. In particular, the orchestrator collects information from SPs about their DNN and the related features before determining the prices of the weights to clear the market equilibrium (Figure 5.1b). Once the sale happens, the original knowledge of the buyer is fused with that newly acquired to increase inference accuracy (Figure 5.1c). More details about the market model are provided next.

5.3.2 Market Model

We employ Fisher’s market [45] to characterize the process of knowledge sharing, wherein the weights of a specific DNN architecture are the goods sold and bought by SPs. Weights have a certain demand that determines their price and the buyers have a certain budget. The benefit obtained by the buyers acquiring these weights is described in terms of a utility function, which expresses the gain in the accuracy from using the acquired weights (Figure 5.1c). The buyers aim at maximizing their utility and the sale occurs multiple times (for instance, periodically) in rounds. An SP can be a buyer in one round and as a seller in another one; however, an SP is either buying or selling weights within a single round of sale. Moreover, weights sold to one seller in a round cannot be sold to another seller in the same round. Still, the same weights can be resold to different buyers in separate rounds.

The weights of DNN type $r$ are denoted by a vector $w_r$. These weights are considered divisible resources, meaning that SPs may trade a subset of them. The size of a given DNN architecture is expressed in terms of its number of layers, also called the capacity of weights. Specifically, the capacity of the weights owned by an SP $\hat{s}$ is given by the vector $c_{\hat{s}}^r = (c_{\hat{s},1}^1, c_{\hat{s},1}^2, \ldots, c_{\hat{s},1}^R)$, where $c_{\hat{s},1}^r$ is the maximum number of layers of a given weight type $r \in R$ owned by SP $\hat{s}$. The base demand of an SP is the minimum amount of weights to improve the accuracy derived from the local knowledge. In detail, the base demand of an SP $s$ for the weights of $\hat{s}$ is denoted as the vector $d_{\hat{s}}^s = \{d_{\hat{s},1}^s, d_{\hat{s},2}^s, \ldots, d_{\hat{s},R}^s\}$ where $d_{\hat{s},r}^s > 0, \forall \hat{s} \in S, r \in R, s \in S_B$ is the demand for weight of type $r$ for SP $s$. Here, $d_{\hat{s}}^s$ is determined by an impartial orchestrator based on the expected accuracy from acquiring the weights, for instance, as in [39, 158]. The weights allocated to each buying SP $s$ from selling SP $\hat{s}$ is denoted by the vector $a_{\hat{s}}^s = (a_{\hat{s},1}^s, a_{\hat{s},2}^s, \ldots, a_{\hat{s},R}^s)$, where $a_{\hat{s},r}^s$ represents the amount of weights for DNN type $r$ allocated to SP $s$ from SP $\hat{s}$. The weights SP $s$ receives from all the other selling SPs are described by the matrix, $A_{\hat{s}} = (a_{\hat{s},1}^1, a_{\hat{s},1}^2, \ldots, a_{\hat{s},R}^1, \ldots, a_{\hat{s},1}^{|S|}, \ldots, a_{\hat{s},R}^{|S|})$, with $A_{\hat{s}} \in \mathbb{R}^{|S| \times R}$.

The utility of an allocation $u_s(A_{\hat{s}})$ describes the buyer’s appraisal for a
weight bundle and is given as follows:

\[ u_s(A_s) = \min \left\{ \sum_{\hat{s} \in S} \min_{r \in R} \frac{a_{s,\hat{s}}}{d_{s,r}}, D_s \right\}, \forall s \]  

(5.1)

where \( D_s \) is the maximum demand of service \( s \). Here, the minimum gain in accuracy for an SP \( s \) from the weights of \( \hat{s} \) among all DNN type \( r \) is \( \min_{r \in R} \frac{a_{s,\hat{s}}}{d_{s,r}} \). This implies that any knowledge transfer should not reduce the accuracy. Moreover, the summation across the sellers denotes the minimum weight gain that can be obtained from all sellers for a given buyer \( s \).

### 5.3.3 Problem Formulation

The market scenario discussed above can be formulated as a utility maximization problem in a Fisher’s market where SPs are the buyers and the sellers and the weights are the goods in a given round. The prices of the weights are expressed through the vector \( p_{\hat{s}} = (p_{\hat{s},1}^{\hat{s}}, p_{\hat{s},2}^{\hat{s}}, \ldots, p_{\hat{s},r}^{\hat{s}}, p_{\hat{s},R}^{\hat{s}}) \), wherein \( p_{\hat{s},r}^{\hat{s}} \) is the price of the weights for DNN type \( r \) at SP \( \hat{s} \). The objective of the market is to determine the optimal price and allocation of the weights to the buyers.

**Problem 2 (Optimal Knowledge Sharing Problem).** The optimal resource allocation for SP \( s \) at price \( p \) maximizes the utility of the allocations as follows:

\[
\begin{align*}
\max_{A_s,u_s} \quad & u_s \\
\text{s.t.} \quad & u_s = \min \left\{ \sum_{\hat{s} \in S} u_{\hat{s}}^{\hat{s},D_s} \right\}, \forall s \\
& u_{\hat{s}}^{\hat{s}} = \min_{r \in R} \frac{a_{\hat{s},\hat{s}}^{\hat{s},r}}{d_{\hat{s},r}}, \forall \hat{s} \\
& \sum_{\hat{s} \in S} \sum_{r \in R} a_{\hat{s},r}^{\hat{s}} p_{\hat{s},r}^{\hat{s}} \leq B_s \\
& a_{\hat{s},r}^{\hat{s}} \geq 0, \forall \hat{s} \in S, r \in R
\end{align*}
\]

(5.2a - 5.2e)

where the constraints\(^1\) in Equation (5.2b) and Equation (5.2c) indicate the utility of the \( s \) and the minimum gain in accuracy for \( s \). The constraint in Equation (5.2d) signifies that the allocation should be within the budget \( B_s \) of each service \( s \), while the one in Equation (5.2e) indicates that the allocation consists of a positive amount of weights.

\(^1\)Capacity constraints on goods do not affect the prices of the resources, therefore they can be removed from the formulation of utility.
5.4 Weight Allocation and Update

Solving the above-mentioned problem means finding a solution that maximizes the utility of each buyer $s$ subject to budget constraints and clears the market. Equivalently, such a solution corresponds to obtaining the prices $\tilde{p}$ and the allocation of weights ($\tilde{A}$) in the condition of market equilibrium [129, 22].

We establish a correspondence between the market equilibrium and an equivalent formulation of Problem 2 that can be solved through convex optimization [129, Chapter 5 and 6]. Specifically, Problem 2 can be rewritten as follows.

**Problem 3** (Extended Eisenberg-Gale Program).

\[
\begin{align*}
\max_{\tilde{A}, \tilde{u}} & \quad \sum_{s \in S} B_s \ln \sum_{\hat{s} \in S} u_{\hat{s}}^s \\
\text{s.t} & \quad u_{\hat{s}}^s \cdot a_{\hat{s},r}^s = a_{s,r}^s, \forall s, \hat{s}, r \\
& \quad \sum_{\hat{s} \in S} u_{\hat{s}}^s \leq D_s, \forall s \in S \\
& \quad \sum_s a_{s,r}^s \leq 1, \forall r \in R, \hat{s} \in S \\
& \quad a_{s,r}^s \geq 0, \forall s, \hat{s}, r
\end{align*}
\]

The formulation above is called the extended Eisenberg-Gale (extended EG or EEG) program. In contrast to the standard EG program, the constraints in Equation (5.3b) and Equation (5.3c) enforce that the utility of buyers does not exceed the maximum demand $D_s$.

An optimal solution to Problem 3 is an exact non-wasteful and economic market equilibrium for a Fisher’s market where buyers either spend all their budget or reach their utility limit. Moreover, all non-wasteful and economic market equilibria captured by Problem 2 are an optimal solution of Problem 3. See Publication V for a more detailed discussion on existence of equilibria and the correspondence between the two problems. In short, solving Problem 3 finds an allocation of weights that maximizes the utility defined in Problem 2 and corresponds to the market equilibrium.

Weight fusion is performed once the equilibrium prices and allocations are obtained. Let $w$ be the local weights of a buyer and $w_i$ the weights acquired from seller $i$, $\forall i$ such that $0 \leq i < n$. The new weights $\tilde{w}$ are then calculated as:

\[
\tilde{w} = w + \frac{1}{n} \sum_{i=1}^{n} \lambda_i (w_i - w)
\]

where $\lambda_i$ are tradeoff parameters such that $0 \leq \lambda_i \leq 1, \forall i \in \{1, \ldots, n\}$. In particular, $\lambda_i = 0$ completely ignores the weights from seller $i$, by keeping...
Algorithm 3: Weight Market Mechanism

Input: A set of buying SPs $s \in S_B$, selling SPs $\hat{s} \in S_S$, broker, and weights for sale $w_r$

Output: Equilibrium prices $\hat{p}$ and allocation $\hat{A}$

// Parameter derivation

1 foreach buyer $s$ and DNN type $r$ do
2     $d_{s^r}$ ← Estimate demand through an API provided by the broker and model in [158] for DNN of type $r$
3     $B_s$ ← Calculate the budget for based on $d_{s^r}$
4     Send $d_{s^r}$ and $B_s$ to broker for sale

5 foreach seller $\hat{s}$ and DNN type $r$ do
6     $p_{\hat{s}^r}$ ← Initial prices of weights for DNN of type $r$
7     Send $p_{\hat{s}^r}$ to the broker for sale

// Market clearance
8 Compute the utilities from received data
9 $\hat{p}, \hat{A}$ ← Compute the equilibrium prices and allocations for buyers and sellers \(\triangleright \text{Eqs. (5.3a) – (5.3e)}\)
10 Send the prices and allocations back to all $s$ and $\hat{s}$
11 All $s$ pay $p_{\hat{s}^r}$ to get $\hat{a}_{\hat{s}^r}$ from $\hat{s}$

// Weight fusion
12 $w_i$ ← $\hat{a}_{\hat{s}^r}$
13 Buyer $s$ fuses $w_i$ into current weight $w$ as $\bar{w}$ \(\triangleright \text{Equation (5.4)}\)

the pre-existing knowledge as it is; and $\lambda_i = 1$, entirely replaces the local knowledge with that of seller $i$.

Algorithm 3 illustrates the entire knowledge sharing process through the weight market mechanism. The algorithm consists of two main phases. In the first phase, the buyers derive their demand based on an estimate of the gain for the given DNN types provided by the broker, update their budget accordingly, and send both of them to the broker. At the same time, the sellers set the initial price for the weights of the given DNN types and send them to the broker. Once all SPs have contacted it, the broker derives the actual utilities, solves Problem 3 and sends the prices as well as the weight allocation back to the SPs (line 10). The allocations are approximated to the nearest integer values to be divisible in practice. Buyers and sellers finally complete the sale (line 11). In the second phase, the buyers update their weights based on bought from the sellers through Equation (5.4). It can be shown that the proposed weight market mechanism satisfies the following economic properties: proportional fairness, defined as the utility of each SP proportional to its budget; and envy freeness, when an SP should not prefer (i.e., envy) the allocation vector and prices of any other SP to its own sharing incentive. Additional details related to the optimality and the economic properties of the weight market mechanism can be found in Publication V.
5.5 Summary of Results

We employ a custom python simulator to evaluate the proposed solution in terms of its economic properties and the accuracy resulting from weight fusion. We consider a reference scenario with 14 sellers (i.e., $|S| = 14$) and 7 buyers (i.e., $|B| = 7$) with three well-known CNN architectures for object recognition: AlexNet (with 12 layers), VGG16 (with 16 layers), and NiN (with 8 layers). The Stanford Cars dataset [93] is used for both training and inference.

We evaluate the discussed market equilibrium and the economic properties of our solution for the EEG program against several benchmarks, namely: Eisenberg-Gale (EG), the standard EG program without the constraints in Problem 3; proportional sharing (PS) [23], which allocates a portion of every resource proportionally to the buyer’s budget; social welfare maximization (SWM) [79], which maximizes the total utility of all the buyers subject to the available supply, without considering the budget constraint; and max-min fairness (MMF), which maximizes the utility of the buyer with the lowest utility subject to the available supply. The selected benchmarks have well-established economic properties and characteristics under market equilibrium. We solve the EG and EEG programs by using the CVXPY modeling language [40] and the MOSEK solver [119] after normalizing the layer count of the considered DNN architectures.

The base demand vector is determined for a distributed intelligence scenario [82, 123], in which the orchestrator predicts the model accuracy expected from the weights by using a gradient boosting machine with regression trees [158].

Figure 5.2 illustrates the economic performance of the considered schemes. Specifically, Figure 5.2a shows the total utility of the buyers as a function of the number of sellers. The total utility saturates in all cases as the number of sellers increases due to restrictions on the demand. EEG and SWM reach the saturation point at a lower number of sellers, with a slight
Knowledge sharing economy

Figure 5.3. Accuracy as a function of the training epoch for knowledge fusion involving one buyer and one seller for different distributions of training data between the buyer and the seller: (a) 30% vs 70%, (b) 40% vs 60%, and (c) 50% vs 50%.

advantage for SWM when there are at most 20 sellers. PS and MMF perform worse; MMF has the lowest total utility due to fair but unbalanced sale of weights, which results in a reduction of knowledge sharing among the SPs. SWM exhibits a total utility which is comparable to that of EEG; however, it causes a higher disparity due to unfair sale. Instead, EEG obtains a total utility with a fair allocation that maximizes the utility of each individual buying SP.

Figure 5.2b shows the proportional fairness (PF) index of the EEG scheme as a function of the number of sellers for the individual buyers. A higher PF index indicates a better proportional allocation with respect to the budget. An allocation consisting of $|S_B|$ buyers with equal budget satisfies PF when the PF of each buyer is greater than or equal to $1/|S_B|$; in the considered case, it should be greater than or equal to $1/7 = 0.1429$. We observe that the PF index is always above 0.2559 even when there are 10 sellers and reaches the value of one for most buyers when there are at least 40 sellers. This confirms that the market equilibrium obtained through EEG satisfies the proportional property.

Figure 5.2c illustrates the envy freeness (EF) index as function of the number of sellers for the considered schemes. The higher the EF index lower the envy felt by an SP. We observe that both EG and EEG achieve a value of one as they satisfy the EF property; the same applies to PF, as a proportional allocation is envy-free by definition. Instead, SWM and MMF are not envy-free unless the number of sellers is large. The results confirm that the proposed EEG scheme for selling weights implies that no SPs prefer (enjoy) the allocation and prices of any other SP.

Figure 5.3 shows the accuracy as a function of the training epoch, for the case of a buyer that received weights from a single seller with EEG, under different distributions of training data. The inference accuracy is shown as a function of the training epoch for different values of the $\lambda$ parameter; $\lambda = 0$ keeps the weights of the buyer as they are, while $\lambda = 1$ replaces the local weights with those of the seller. The FedAvg scheme
employed in federated learning is also reported for comparison purposes. In particular, Figure 5.3a illustrates the accuracy when the seller has 30% of the total training data. We observe that the accuracy more than doubles in this scenario. Higher values of $\lambda$ have a more significant impact on the accuracy when the difference between the training data at the buyer and at the seller is smaller shown by Figures 5.3b and 5.3c. The difference in the accuracy for $\lambda = 0$ and $\lambda = 1$ decreases as the difference in the amount of training data at the buyer and at the seller decreases. FedAvg obtains an accuracy similar to using weight fusion with $\lambda = 0.5$, as it averages all the weights involved in the distributed training process (i.e., of both sellers and buyers). In contrast, the weight fusion approach employed here does not require exchanging weights multiple times during training, but leverages the allocation of weights found by the broker.

5.6 Conclusion

This chapter discussed knowledge sharing between multiple providers of AI services. Specifically, we presented an economic model based on Fisher's market to buy and sell weights as well as a weight fusion technique to incorporate the acquired knowledge. The proposed approach achieves a market equilibrium and satisfies important economic properties, including envy freeness and proportional allocation. Moreover, the weight fusion mechanism was shown to be effective in improving local accuracy, under different distributions of training data at the buyer and the sellers. A promising future work is to explore the dynamics that result from carrying out trading periodically over time. Other research directions include using non-linear utility functions, simultaneous sales by sellers and buyers through an Arrow–Debreu market, and alternative formulations of the knowledge sharing problem, including auction mechanisms.
6. Conclusion

“In literature and in life we ultimately pursue, not conclusions, but beginnings.”

Sam Tanenhaus, Literature Unbound

Rapid progress in information and communication technologies has given rise to emerging applications, including different services leveraging AI and ML. The heterogeneity of these applications as well as their resource requirements has raised several challenges that remain yet to be fully addressed. This dissertation considered some of them, including distributed and heterogeneous resource management for AI-based services in IoT scenarios. Specifically, we proposed efficient computing and communication solutions to manage the heterogeneous and ever-growing resource demands of emerging applications.

First, we focused on carrying out DNN inference acceleration in the IoT. Specifically, we showed how to leverage edge and fog computing to provide a fast and scalable DNN inference for resource-constrained embedded devices. To this end, we devised two distributed techniques that collaboratively partition and offload computations under dynamic network conditions to minimize the DNN inference time. Extensive simulations in realistic settings demonstrated that the algorithms achieve a significant reduction in inference time, better load balance, and fair offloading compared to the state of the art.

Second, we targeted improving the execution time of inference involving sparse DNNs on GPUs. Specifically, we streamlined the performance of SpMV computations, which are widely used in several scientific applications as well as with pruned DNNs. To this end, we proposed DIESEL, a tool leveraging DL to predict and execute the best performing SpMV kernel for a given matrix. The selection was done based on a set of features that were selected through rigorous empirical and analytical factors. The results obtained from the experiments showed that our tool provides the best performance among alternative solutions in the state of the art across different performance metrics.
Third, we focused on efficient and economic network resource utilization under a target service quality. More specifically, we leveraged network slicing and Fog-RANs to improve resource utilization as well as the profits of multiple SPs delivering generic uplink and downlink services in 5G networks – eMBB, URLLC, or mMTC. To this end, we proposed a hierarchical resource scheduling mechanism named 2L-MRA to jointly allocate multiple Fog-RAN resources to slices in two stages: multi-resource allocation to slices at fog nodes over a given time window and a slice-specific resource allocation at each fog node to users with a much shorter time scale. 2L-MRA satisfies several economic properties, including Pareto optimality, envy freeness, and sharing incentive. Extensive simulations in realistic scenarios have demonstrated that 2L-MRA significantly increases utility in varying network conditions, thereby resulting in higher utilization and better QoS.

Finally, we focused on improving the accuracy of DNNs in addition to their inference time. More specifically, we proposed a market model that incentivizes different SPs to trade their existing knowledge and fuse it to increase their model accuracy. To this end, we devised a model based on Fisher’s market for optimal trading as well as a weight fusion technique to combine the acquired knowledge with the one available locally. The proposed approach achieves a market equilibrium and satisfies important economic properties, including envy freeness and proportional allocation. Extensive simulations showed a significant increase in the inference accuracy compared to the original DNN.

6.1 Future Work

This section describes the limitations of the solutions presented in this dissertation as well as directions for extending them.

We focused on carrying out DNN inference acceleration for resource-constrained devices. However, we have evaluated our solutions only analytically or by simulation, without running them on real devices. A promising future work is indeed represented by evaluating the algorithms in edge or fog testbeds with heterogeneous resource-constrained devices. Other research directions include addressing DNN inference for areas with challenging connectivity due to poor channel conditions and (or) device mobility. Additionally, another interesting aspect includes addressing distributed DNN learning instead of inference.

We addressed efficient computation of sparse/pruned DNNs by proposing the DIESEL tool to select the best storage scheme and the associated algorithm for SpMV. However, DIESEL only supports five storage techniques and execution on GPUs. A promising future work is extending DIESEL with additional iterative methods and SpMV kernels. Other research oppor-
tunities involve considering SpMV support for heterogeneous computing architectures including multiple integrated cores and field programmable gate arrays.

We leveraged slicing and Fog-RANs to improve the efficiency of 5G networks in terms of both resource utilization and economic profit. A promising future work is to consider how the pricing of different resource types affects the economics of multi-resource allocation with several SPs. Another promising direction is given by the design of techniques based on machine learning to allocate heterogeneous resources according to different application requirements.

We discussed a market model that leverages a model based on Fisher’s market for optimal trading. A promising future work is to explore the dynamics that result from carrying out trading periodically over time. Other research directions include alternative formulations of the knowledge-sharing problem, including auction mechanisms.
References


References


[145] F. Sattler, S. Wiedemann, K.-R. Muller, and W. Samek. Robust and communication-efficient federated learning from non-i.i.d. data. *IEEE Transactions on Neural Networks Learning Sys.*, 31(9):3400–3413, 2020.


Errata

Publication I

Problem 2 in second paragraph page 4 should be Problem 1