University of Kentucky
UKnowledge

Theses and Dissertations--Computer Science

Computer Science

2022

Smart Decision-Making via Edge Intelligence for Smart Cities

Nathaniel Hudson
University of Kentucky, nchudson88@gmail.com
Author ORCID Identifier:
https://orcid.org/0000-0001-7474-2689
Digital Object Identifier: https://doi.org/10.13023/etd.2022.82

Right click to open a feedback form in a new tab to let us know how this document benefits you.

Recommended Citation

Hudson, Nathaniel, "Smart Decision-Making via Edge Intelligence for Smart Cities" (2022). *Theses and Dissertations--Computer Science*. 117.
https://uknowledge.uky.edu/cs_etds/117

This Doctoral Dissertation is brought to you for free and open access by the Computer Science at UKnowledge. It has been accepted for inclusion in Theses and Dissertations--Computer Science by an authorized administrator of UKnowledge. For more information, please contact UKnowledge@lsv.uky.edu.

STUDENT AGREEMENT:

I represent that my thesis or dissertation and abstract are my original work. Proper attribution has been given to all outside sources. I understand that I am solely responsible for obtaining any needed copyright permissions. I have obtained needed written permission statement(s) from the owner(s) of each third-party copyrighted matter to be included in my work, allowing electronic distribution (if such use is not permitted by the fair use doctrine) which will be submitted to UKnowledge as Additional File.

I hereby grant to The University of Kentucky and its agents the irrevocable, non-exclusive, and royalty-free license to archive and make accessible my work in whole or in part in all forms of media, now or hereafter known. I agree that the document mentioned above may be made available immediately for worldwide access unless an embargo applies.

I retain all other ownership rights to the copyright of my work. I also retain the right to use in future works (such as articles or books) all or part of my work. I understand that I am free to register the copyright to my work.

REVIEW, APPROVAL AND ACCEPTANCE

The document mentioned above has been reviewed and accepted by the student's advisor, on behalf of the advisory committee, and by the Director of Graduate Studies (DGS), on behalf of the program; we verify that this is the final, approved version of the student's thesis including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

Nathaniel Hudson, Student

Dr. Hana Khamfroush, Major Professor

Dr. Simone Silvestri, Director of Graduate Studies

Smart Decision-Making via Edge Intelligence for Smart Cities

DISSERTATION

A dissertation submitted in partial
fulfillment of the requirements for the
degree of Doctor of Philosophy in the
College of Engineering at the University
of Kentucky

By
Nathaniel Hudson
Lexington, Kentucky

Director: Dr. Hana Khamfroush
Assistant Professor of Computer Science
Lexington, Kentucky 2022

Copyright© Nathaniel Hudson 2022

ABSTRACT OF DISSERTATION

Smart Decision-Making via Edge Intelligence for Smart Cities

Smart cities are an ambitious vision for future urban environments. The ultimate aim of smart cities is to use modern technology to optimize city resources and operations while improving overall quality-of-life of its citizens. Realizing this ambitious vision will require embracing advancements in information communication technology, data analysis, and other technologies. Because smart cities naturally produce vast amounts of data, recent artificial intelligence (AI) techniques are of interest due to their ability to transform raw data into insightful knowledge to inform decisions (e.g., using live road traffic data to control traffic lights based on current traffic conditions). However, training and providing these AI applications is non-trivial and will require sufficient computing resources. Traditionally, cloud computing infrastructure have been used to process computationally intensive tasks; however, due to the time-sensitivity of many of these smart city applications, novel computing hardware/technologies are required. The recent advent of edge computing provides a promising computing infrastructure to support the needs of the smart cities of tomorrow. Edge computing pushes compute resources close to end users to provide reduced latency and improved scalability — making it a viable candidate to support smart cities. However, it comes with hardware limitations that are necessary to consider.

This thesis explores the use of the edge computing paradigm for smart city applications and how to make efficient, smart decisions related to their available resources. This is done while considering the quality-of-service provided to end users. This work can be seen as four parts. First, this work touches on how to optimally place and serve AI-based applications on edge computing infrastructure to maximize quality-of-service to end users. This is cast as an optimization problem and solved with efficient algorithms that approximate the optimal solution. Second, this work investigates the applicability of compression techniques to reduce offloading costs for AI-based applications in edge

computing systems. Finally, this thesis then demonstrate how edge computing can support AI-based solutions for smart city applications, namely, smart energy and smart traffic. These applications are approached using the recent paradigm of federated learning.

The contributions of this thesis include the design of novel algorithms and system design strategies for placement and scheduling of AI-based services on edge computing systems, formal formulation for trade-offs between delivered AI model performance and latency, compression for offloading decisions for communication reductions, and evaluation of federated learning-based approaches for smart city applications.

KEYWORDS: smart cities, edge computing, artificial intelligence, edge intelligence, information communication technology

Nathaniel Hudson

April 30, 2022

Smart Decision-Making via Edge Intelligence for Smart Cities

By

Nathaniel Hudson

Dr. Hana Khamfroush

Director of Dissertation

Dr. Simone Silvestri

Director of Graduate Studies

April 30, 2022

Date

DEDICATION

This work is dedicated to my mother, Emma Hudson, and my father, Clinton Hudson. Both of their lifelong support made the completion of this thesis possible.

ACKNOWLEDGMENTS

There are countless people that I would like to take the time to acknowledge for helping me along the way towards completing this thesis. First and foremost is my advisor, Dr. Hana Khamfroush, whose mentorship has been deeply invaluable and irreplaceable. She has consistently challenged me to grow as a researcher and a person. I am deeply thankful to have had her serve as my advisor. Further, I am thankful for each my committee members: Dr. Stephen Borgatti, Dr. Judy Goldsmith, Dr. Brent Harrison, and Dr. Miroslaw Truszczynski. Finally, I would like to acknowledge and thank both the National Science Foundation under grant no. CSR-1948387 and Cisco Systems Inc. under the research grant number 1215519250 for funding portions of this thesis.

TABLE OF CONTENTS

Acknowledgments . . . iii

List of Tables . . . viii

List of Figures . . . ix

Chapter 1 Introduction . . . 1
1.1 Contribution . . . 4
1.2 Roadmap of the Thesis . . . 5

Chapter 2 QoS-Aware Placement of Deep Learning Services on the Edge with Multiple Service Implementations . . . 6
2.1 Introduction . . . 7
2.2 Related Works . . . 8
2.3 System Model . . . 10
2.3.1 System Architecture Definition . . . 10
2.3.2 Quality-of-Service (QoS) Definition . . . 11
2.4 Problem Definition . . . 13
2.4.1 PIES Formulation . . . 14
2.4.2 PIES Problem Complexity & Properties . . . 14
2.5 Efficient Algorithm Design . . . 18
2.5.1 Scheduling Sub-Problem . . . 19
2.5.2 Placement Sub-Problem . . . 19
2.6 Experimental Design & Results . . . 22
2.6.1 Baselines . . . 22
2.6.2 Numerical Simulations . . . 22
2.6.3 Real-World Implementation . . . 24
2.7 Conclusions . . . 25

Chapter 3 Joint Compression and Offloading Decisions for Deep Learning Services in 3-Tier Edge Systems . . . 26

3.1 Introduction . . . 27
3.2 Related Work . . . 28
3.3 System Design & CODE Problem Definition . . . 29
3.3.1 3-Tier System Architecture . . . 30
3.3.2 Compression and Machine Learning Accuracy . . . 30
3.3.3 CODE Problem Definition . . . 31
3.3.4 Defining Average Accuracy & Latency . . . 32
3.4 Problem Decomposition & Heuristic Design . . . 34
3.4.1 Efficient Algorithm Design . . . 35
3.5 Experimental Design . . . 37
3.5.1 Benchmark Algorithms/Heuristics . . . 38
3.5.2 Environment Setup . . . 39
3.6 Results . . . 40
3.6.1 Impact of $A_{\min}$ & $L_{\max}$ on the Objective Value . . . 41
3.6.2 Impact of α on Compression Decisions ($\mathbf{s}$) and the Objective Value . . . 41
3.6.3 Impact of Hardware Resources . . . 43
3.7 Conclusions . . . 43

Chapter 4 A Framework for Edge Intelligent Smart Distribution Grids via Federated Learning 44
4.1 Introduction . . . 45
4.2 Smart Distribution Grids & Advanced Metering Infrastructure . . . 46
4.2.1 AMI Overview and Requirements . . . 47
4.2.2 AMI Functions and NILM . . . 48
4.3 Edge Intelligence for Smart Grids . . . 49
4.3.1 EC Overview and Architecture . . . 49
4.3.2 Edge Intelligence . . . 51
4.3.3 Federated Learning . . . 51
4.4 FL-Based NILM Model . . . 53
4.4.1 Deep Neural Network Model Design . . . 53
4.4.2 2-Tier & 3-Tier Federated Learning . . . 54
4.5 Experiment Design . . . 55
4.5.1 Data Description . . . 55

4.5.2 Communication Model . . . 56
4.6 Results & Discussion . . . 57
4.6.1 Network Load Analysis . . . 57
4.6.2 Trained Model Performance . . . 59
4.7 Conclusions & Future Directions . . . 60

Chapter 5 Smart Edge-Enabled Traffic Lights Using Federated Reinforcement Learning for Reward-Communication Trade-Off . . . 61
5.1 Introduction . . . 62
5.2 System Description . . . 64
5.3 Proposed SEAL Model Definition . . . 65
5.3.1 Action Space . . . 66
5.3.2 State Space . . . 67
5.3.3 Reward Function . . . 68
5.3.4 Communication Model . . . 69
5.4 Training Algorithms . . . 69
5.4.1 Centralized Training . . . 70
5.4.2 Decentralized Training . . . 71
5.4.3 Federated Training . . . 71
5.5 Experiment Design . . . 73
5.5.1 Considered Road Network Topologies . . . 73
5.5.2 Training Parameters . . . 74
5.6 Results & Discussion . . . 74
5.6.1 Reward Evaluation During Training . . . 75
5.6.2 Communication Cost Evaluation During Training . . . 75
5.6.3 Trained Policy Network Performance . . . 77
5.7 Related Works . . . 78
5.8 Conclusions . . . 79

Chapter 6 Conclusions & Final Remarks . . . 80
6.1 Findings . . . 80
6.2 Future Work . . . 81

Bibliography 83

Vita 100

LIST OF TABLES

2.1 Image classifications models used for the real-world implementation with model accuracy metrics and average computational delay from evaluation with ImageNet 2012 data. 25

3.1 The average objective value of all algorithms based on different α values (note that "-" means that algorithm did not find valid solution for that case.) 42

LIST OF FIGURES

1.1 Our considered edge computing system model. The topmost layer, central cloud, has expansive hardware resources but is further away from the end users — resulting in large delays for end users. The middle layer is the layer of edge clouds which are placed closer to the user. The bottom layer is populated by end users that submit requests to the system. The central cloud and edge cloud collaboratively address user requests following some protocol or policy. 3

2.1 3-tier MEC architecture where an edge cloud has two image classification models to serve requests for that service. 8

2.2 Example auxiliary multigraph representing PIES scheduling sub-problem. Links between user/service nodes correspond to an model implementation for a user's requested service placed on the user's covering edge. Float values correspond with QoS weights. Solid links compose the maximum spanning tree for optimal model scheduling. 16

2.3 **Validation Test.** For these experiments, we consider $|\mathcal{U}| = [50, 100, 150, 200, 250]$, with 10 trials each. We perform this validation test to confirm the efficacy of EGP relative to the optimal solution and the approximation algorithm, AGP. 22

2.4 **Numerical Results.** Experiments with more user requests $|\mathcal{U}| = [100, 200, \cdots, 1000]$, with 100 trials each. 23

2.5 **Real world implementation.** (5a) QoS distribution achieved by each of the algorithms. (5b) Average placement decision for each image classification model across 100 trials. 25

3.1 Illustrative overview of our considered 3-tier architecture. 29

3.2 CDF of Objectives (OPT and OPT_2 are overlapping and are the leftmost curves) . . . 40

3.3 Impact of α on the proportion of served requests by each layer 40

3.4 Impact of α on the compression decision 41

3.5 Impact of edge servers and cloud server CPU on objective value 42

4.1 An overview of EI-AMI architecture. AMI infrastructure (in yellow) are resources deployed specifically for the EI-AMI architecture and is a subcomponent of the distribution grid. General infrastructure (in purple) are EC resources deployed for general use but can be leveraged through communication channels to supplement the AMI infrastructure. 49
4.2 The network load for learning using centralized training, 2-tier FL, and 3-tier FL with Eqs. (4.1), (4.2), (4.3) (respectively). We consider the fraction of participating homes for 2-tier and 3-tier FL to be $C = 1$. Number of hops from smart homes to AMI-Edge nodes and AMI-Edge nodes to the central utility server uniformly range from $[1, 10]$. . 58
4.3 Loss performance: (a) Comparison between the training loss over epochs (Centralized) and federated aggregation rounds (2-tier and 3-tier FL). (b) The CDF distribution of the testing loss after the models have been trained — which shows a tight similarity in performance. 58

5.1 Example of our traffic system where traffic lights communicate with an edge-enabled roadside unit (Edge-RSU). 64
5.2 Example traffic light action transition graph. Consider the given traffic light k's current phase state is `GGrGGR`. If the action $a_k^t = 1$ at time-step t, then the phase state for k will transition to `yyryyr` if sufficient time has elapsed since its last transition. Otherwise, its phase state remains the same, unless too much time has elapsed since its last change. 66
5.3 Training approaches considered for solving SEAL. 70
5.4 Example Grid-3 $\times$ 3 road network with heterogeneous intersection types. Note that the number of lanes increase as roads are more central. 74
5.5 Learning curves with each training approach on each road network. 75
5.6 Communication cost (i.e., data size in bytes) transmitted during training time under each training strategy for each road network. 76
5.7 Evaluation of trained policy networks on each road network using trip metrics, namely Travel Time (top row) and Waiting Time (bottom row). We compare the results to a Pre-Timed phase transition model as a baseline. Results confirm the RL-based solutions generally outperform the baseline. 76

1

Introduction

One of the most noteworthy arcs of the 21st century has been the explosive growth seen in the usage and deployment of technology. More specifically, the advancement of *information and communications technology* (ICT) has been the foundation for many of the innovations seen over this century thus far. The wide-spanning adoption of the Internet for social media, e-commerce, news consumption, and more has redefined much of our everyday lives and industries across the world. Due to the advancements of ICT, there are many exciting opportunities for what is yet to come. One longstanding opportunity for the future is the vision of the *smart city*.

There are countless different ideas and definitions for what exactly qualifies as a smart city. A more general definition, provided by Bakıcı et al. in [1], defines smart cities as being high-tech cities that serve to connect people, information, and city-wide infrastructure and resources using advances in technology to improve sustainability, innovation, and overall quality-of-life. The extent to which how pervasive (or intrusive) technologies are incorporated to accomplish such goals is often a point of contention in the literature [2]. However, the many different visions for what define smart cities all generally involve heavy utilization of technology to improve the maintenance of cities to further improve and/or optimize city-specific operations for different stakeholders (e.g., citizens, the environment).

Realizing the ambitious vision of smart cities requires seamlessly making decisions for a host of different applications and domains with minimal sacrifice with respect to performance and resource costs. Some applications of interest for smart cities include, but are not limited to, smart traffic light control, smart information spread (or advertisement), smart energy distribution, smart waste management, etc. Ideally, data generated in urban environments can be collected in smart cities using

ICT-powered infrastructure to train and inform models designed for these applications to make real-time decisions. For instance, insistent collection of traffic congestion data from *roadside units* (RSU) and other sources can be sent to ICT infrastructure to train machine learning models how to make optimal decisions related to traffic lights to reduce congestion. This approach is becoming more within reach given the advances in *vehicular ad hoc networks* (VANET) [3, 4, 5] and other Internet-of-Vehicles (IoV) systems [6, 7, 8].

Making smart decisions in a timely fashion will require a powerful pipeline ready to support intense amounts of data collection, analytics, and adaptive decision-making. For instance, video feeds at road intersections — to empower models for smart traffic light control — are large in size (in terms of bytes) and will require automated processes to make meaningful decisions in real-time. However, video cameras that provide such video feeds will be unable, on their own, to perform such analytics and decision-making. Such tasks would require more computational resources than would be available at video cameras and other similar sensors that can be found in urban environments. Thus, for such applications, smart cities will need to be able to provide heavy computational power on-demand, even for devices that do not have such resources locally. Applications (e.g., big data analytics, video classification) all require large amounts of compute resources in order to adequately process. Because Internet-of-Things (IoT) devices like smartphones are often ill-equipped to meet the needs of such processes [9, 10], a standard approach for providing these services on these devices is to rely on remote resources.

A common solution for this problem of leveraging remote resources is known as *cloud computing* [11, 12], where large and remote computational resources can be accessed to serve service requests of virtually any kind of device. The advent of the cloud has seen rapid growth in recent years alongside the advancements of communication and storage technologies that make up the basis of the Internet [13]. Despite the promise of cloud computing, there are notable and severe limitations that need to be considered — particularly when considering the realization of smart cities [14, 15, 16]. First, the cloud being a remote resource depreciates its ability to provide optimal latency for received requests. Many applications in the interest of smart cities are critical and delay-sensitive, meaning that we need to ensure that cyber infrastructure supporting smart cities provides optimal latency. Second, the cloud serving as a centralized data storage entity also restricts the ability to provide data privacy due to entities from unknown sources also relying on the same cloud resources. Third, having critical infrastructure for smart cities rely on a centralized entity raises concerns of reliability. With the rapidly growing number of IoT devices around the world, it may not always be preferable

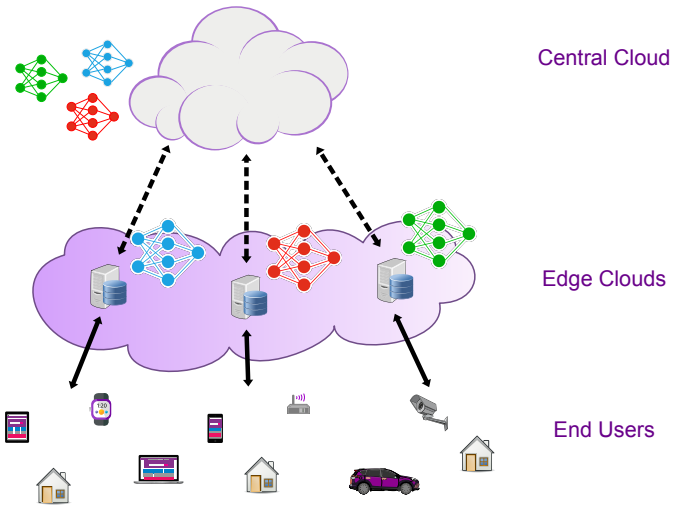

Figure 1.1: Our considered edge computing system model. The topmost layer, central cloud, has expansive hardware resources but is further away from the end users — resulting in large delays for end users. The middle layer is the layer of edge clouds which are placed closer to the user. The bottom layer is populated by end users that submit requests to the system. The central cloud and edge cloud collaboratively address user requests following some protocol or policy.

to rely on the same system other users outside the city are using.

In light of these possible limitations of cloud computing, a more recent paradigm has sparked a lot of excitement. This paradigm is known as *edge computing* [15, 17]. Also commonly referred to as fog [18, 19] and micro-cloud computing [20, 21], the edge computing framework serves as the next iteration of the cloud, where compute and other resources are pushed to the edge of the network and closer to end users to improve latency and overall Quality-of-Service (QoS). The general idea is that when a user wishes to perform some computationally-intensive task (e.g., smart voice assistant), the user can submit a request for a service to be performed on the edge computing infrastructure. These edge infrastructure can be thought of as micro-clouds with their own hardware to handle service requests to provide a response to users. Hereafter, these edge infrastructure devices will be referred to as "edge clouds". The physical proximity of edge clouds to the end users provides for improved latency. Additionally, the distributed nature of edge clouds (in comparison to a centralized cloud) can provide opportunities for building more reliable, robust, and secure ICT infrastructure for smart cities. Additionally, edge computing can interact with conventional, centralized cloud infrastructure to collaboratively manage user requests. Due to the various architectures that can be composed for edge computing environments, it is crucial that the architecture we consider is clarified. In this work, we consider (unless specified otherwise) a 3-tier edge computing architecture. The topmost tier is the central cloud; the middle tier is the layer of edge clouds; and the bottommost tier is the layer of

end users that submit service requests to the system. A visual overview of this architecture can be found in Figure 1.1.

Growing popularity in edge computing has generated much interest in the possibility of deploying complex models for machine intelligence on the edge. More succinctly, this idea is commonly referred to as *edge intelligence* [22, 23]. Examples of models that can be used to provide machine intelligence on the edge include (but are not limited to) deep learning models, unsupervised clustering models, and regression models. Deep learning models (e.g., deep neural networks [24], convolutional neural networks [25], recurrent neural networks [26]) are becoming increasingly popular due to their relevance for a variety of applications and domains from energy disaggregation, facial recognition, robotics, and natural language understanding [27, 28, 29, 30].

Edge intelligence is an interesting and unique area of study because it requires the consideration of the advantages and limitations of edge computing systems. For instance, edge ICT infrastructure being physically close to where data are generated presents an interesting and timely opportunity to train deep neural networks. However, the hardware resources of edge clouds (e.g., communication, compute, and storage resources) are relatively constrained in comparison to a large, centralized cloud. As such, the exorbitant computational cost associated with training these complex deep learning models to provide intelligence is a valid concern. With that said, there is much room to understand how to optimally deploy, train, and use these models on the edge without incurring excessively large amounts of resource costs.

1.1 Contribution

This thesis studies how edge intelligence can be used to support applications relevant for realizing the ambitious vision of smart cities. This is done by considering an edge computing platform for providing and training edge intelligence solutions on the edge. Contributions of this thesis include novel algorithms, system design strategies, optimization formulations for trade-offs of such systems, etc. This thesis approaches this broad by topic by first discussing placement and scheduling strategies for edge intelligence services. Then, compression techniques are considered for reducing communication costs associated for offloading requests in such edge computing infrastructure. Finally, this thesis then demonstrates the potential of edge intelligence solutions for smart cities by considering two relevant applications, namely, *smart energy distribution* and *smart traffic control*.

1.2 Roadmap of the Thesis

This thesis begins in Chapter 2 by first considering decisions for where to place edge intelligence services and how to schedule said services to process user requests to maximize total Quality-of-Service (QoS). This chapter considers QoS to be a function of both accuracy and latency provided to user requests. Chapter 3 then considers the application of compression techniques to improve implicit accuracy-latency trade-offs when scheduling user requests. The final two chapters consider two smart applications. Chapter 4 focuses on smart grid applications and the role edge computing infrastructure can serve for smart grid systems. More specifically, the recent advent of federated learning is applied to the problem of Non-Intrusive Load Monitoring (NILM) to improve communication costs for training edge intelligence applications. Finally, Chapter 5 studies how federated reinforcement learning for the application of smart traffic light control can be used to improve road conditions while reducing associated communication costs for smart cities.

Copyright© Nathaniel Hudson, 2022.

2

QoS-Aware Placement of Deep Learning Services on the Edge with Multiple Service Implementations

©2021 IEEE. Reprinted with permission from Nathaniel Hudson, "QoS-Aware Placement of Deep Learning Services on the Edge with Multiple Service Implementations," 2021 International Conference on Computer Communications and Networks (ICCCN) Big Data and Machine Learning for Networking (BDMLN) Workshop, July 2021,
DOI: 10.1109/ICCCN52240.2021.9522156

Mobile edge computing pushes computationally-intensive services closer to the user to provide reduced delay due to physical proximity. This has led many to consider deploying deep learning models on the edge — commonly known as edge intelligence (EI). EI services can have many model implementations that provide different QoS. For instance, one model can perform inference faster than another (thus reducing latency) while achieving less accuracy when evaluated. In this chapter, we study joint service placement and model scheduling of EI services with the goal to maximize Quality-of-Servcice (QoS) for end users where EI services have multiple implementations to serve user requests, each with varying costs and QoS benefits. We cast the problem as an integer linear program and prove that it is NP-*hard*. We then prove the objective is equivalent to maximizing a monotone increasing, submodular set function and thus can be solved greedily while maintaining a $(1-1/e)$-approximation guarantee. We then propose two greedy algorithms: one that theoretically guarantees this approximation and another that empirically matches its performance with greater efficiency. Finally, we thoroughly evaluate the proposed algorithm for making placement and scheduling decisions in both synthetic and real-world scenarios against the optimal solution

and some baselines. In the real-world case, we consider real machine learning models using the ImageNet 2012 data-set for requests. Our numerical experiments empirically show that our more efficient greedy algorithm is able to approximate the optimal solution with a 0.904 approximation on average, while the next closest baseline achieves a 0.607 approximation on average.

2.1 Introduction

The growth of the Internet has given birth to the advent of the Internet-of-Things (IoT). This ecosystem consists of countless different devices, or things (e.g., sensors, home appliances), that can seamlessly communicate with one another. More importantly, these devices also serve to generate/collect data. In order to acquire meaningful information from these data, they must first be processed. The scale of the IoT poses a problem for processing these data in a timely fashion through a conventional, centralized approach (e.g., cloud computing). As such, a promising framework to approach this problem has been that of *mobile edge computing* (MEC) [31, 15, 32, 33].

The MEC framework considers the deployment of *edge clouds* (or *edge servers*) that provide communication, compute, and storage resources closer to the end user devices to ameliorate latency incurred from physical distance. This physical proximity allows for more immediate and timely data processing for nearby devices in IoT. However, MEC is not without challenges. The hardware resources available at an individual edge cloud pales in comparison to that available at the far away central cloud server. As such, decisions related to how these resources are spent must be optimized. Regardless, MEC remains a promising framework for performing timely data processing for IoT devices, such as smart sensors.

The popularity of *machine learning* (ML) for performing the task of data processing has skyrocketed in recent years. ML has been shown to be capable of achieving remarkable accuracy for complex tasks (e.g., image classification, video classification, speech-to-text). Due to the flexibility and performance of ML technologies, deploying such models on the edge to process IoT data is promising. Thus, the notion of *edge intelligence* (EI) has gained prominence. A notable feature of EI services, such as image classification, is that several different EI architectures (i.e., deep neural networks) can be implemented to perform inference for some input for a service. These different architectures can have varying trade-offs in terms of the time they take to perform inference, the size of input data they require, and how accurate they are in practice when performing inference. Given these observations, in this work, we adapt the well-studied problem of *service placement* in MEC

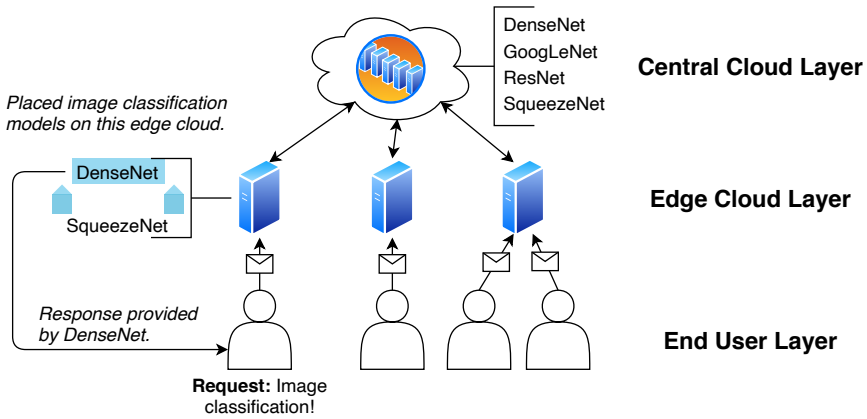

Figure 2.1: 3-tier MEC architecture where an edge cloud has two image classification models to serve requests for that service.

to consider EI services with different model implementations. Most notably, this problem aims to maximize the Quality-of-Service (QoS) provided by the edge when services are allowed to have several different model implementations to serve user requests.

To the best of our knowledge, this is the first EI service placement work that considers each service to have multiple implementations. We summarize our contributions as follows:

- Introduce the *Placement of Intelligent Edge Services* (PIES) problem for optimal placement and scheduling of EI services with multiple implementations to maximize QoS provided by the edge.
- Prove that the PIES problem is NP-*hard* and that its objective maximizes a monotone increasing, submodular set function under matroid constraints.
- Propose two greedy algorithms: one with a $(1 - 1/e)$-approximation guarantee and another more efficient greedy algorithm that empirically matches this approximation algorithm with much greater efficiency.
- Our empirical results show that both greedy algorithms outperform their minimal approximation guarantee and each achieve an approximation of roughly 0.9 on average.

2.2 Related Works

Service Placement. The problem of deciding which services to place on edge clouds in MEC is commonly known as the *service placement* problem. Many works study the problem with the goal

to optimize resource utilization, energy consumption [34, 35], and Quality-of-Service (QoS) [36, 37]. Some works consider a static placement decision where the placement decision is made in a single shot [38, 39]. Other works focus on dynamic placement where placement decisions are made over some timespan [21, 40]. The case where edge clouds can share their resources with one another to collaboratively serve user requests has been studied by He et al. in [41]. The existence of mobile end users has encouraged research studying placement with service migration, where a service processing a request migrates between edge clouds [42]. For a recent and comprehensive survey on service placement, please refer to [43]. *Quality-of-Service* (QoS) is very domain specific. Gao et al. in [36] study maximizing QoS, defined as a function of latency, through a joint decision problem for both service placement and network selection where users can be served by more than one edge cloud. Skarlat et al. define FSPP, a QoS-aware service placement problem, as an ILP where QoS is defined as a function of application deadlines — using the Gurobi solver to provide the optimal solution [44]. Yousefpour et al. in [45] propose a dynamic fog computing framework, called FogPlan, for dynamic placement and release of services on fog nodes in IoT infrastructure. The authors consider QoS-aware service placement where QoS is considered exclusively with respect to delay latency and user's delay tolerance. Wang et al. in [34] study a similar problem wherein they focus on the placement of virtual machines for software-defined data centers to maximize energy efficiency and QoS. An application placement policy using a fuzzy logic-based approach is proposed by Mahmud et al. in [37] that maximizes QoE. In this work, the authors consider QoE under a fuzzy framework comprised of sets for access rate, required resources, and processing times. Farhadi et al. in [46] study service placement and request scheduling on MEC systems for data-intensive applications. They pose the problem of service placement as a set optimization problem and provide an algorithm that demonstrates an approximation bound on optimal solutions.

Edge Intelligence. The advent of pushing machine learning services to the edge led to the established field of *edge intelligence* (EI) [47]. Due to the limited resource capacities of MEC environments, a central focus of EI is the design of models that are less costly in terms of resources to run [48]. One proposed solution is to simply prune the elements comprising the deep neural network for the EI service (e.g., remove the number of neurons/units or entire layers) [49, 50, 51]. Another proposed idea is to consider a EI model's architecture being *split* across different tiers of the MEC architecture (e.g., one half is run on the edge and the other on the central cloud) [52]. Other works have studied optimizing the QoS provided by deep learning EI models on the edge. Zhao et al. in [53] study the trade-off between accuracy and latency for offloading decisions regarding deep learning

services for provided optimal QoS on the edge through compression techniques. Hosseinzadeh et al. in [54] study the related problem of offloading and request scheduling in edge systems to maximize QoS for deep learning models under the assumption that placement of these models has been done a priori. There are few works that have formally studied the service placement problem specifically for EI models. Recently in 2021, Zehong, Bi, and Zhang study the problem of optimally placing EI services with the objective of optimizing energy consumption and completion time [55].

Our work departs from the literature in that we focus on EI service placement where each service can have multiple implementations. To the best of our knowledge from surveying the literature, this is the first work to do so. A similar work by Hung et al. in [56] considers multiple implementations of services. However, the focus of that work is on optimal query scheduling of video processing tasks rather than placement of services on edge clouds.

2.3 System Model

2.3.1 System Architecture Definition

We consider a 3-tier MEC architecture consisting of a central cloud, edge clouds, and end users (see Figure 2.1). The central cloud hosts all model implementations for each service in the environment. However, the objective of this work is to maximize expected QoS provided by the edge clouds and thus we do not closely consider the central cloud. For this work, we focus on three aspects of this environment: edge clouds, end user requests, and available EI service models. For simplicity, we consider requests processed by the cloud to be dropped.

Edge Clouds. We denote the set of edge clouds $\mathcal{E} = \{1, \cdots, E\}$ where E is the number of edge clouds. We consider edge clouds to be deployed computational devices that are associated with a wireless access point to connect to users. Each edge cloud is equipped with hardware resources to process the requests provided by end users. Specifically, we consider each edge cloud $e \in \mathcal{E}$ to have resource capacities K_e, W_e, and R_e for communication, computation, and storage capacities, respectively. For simplicity, we do not consider the possibility of edge clouds collaborating to serve user requests. Thus, either a user's request is served by the user's covering edge cloud or is offloaded to the central cloud (dropped).

User Requests. We denote the set of user requests as $\mathcal{U} = \{1, \cdots, U\}$ where U is the number of user requests. For simplicity, we consider each user to make a single service request. For users that

request multiple services, we represent this as separate user requests altogether. Each user is covered by some edge cloud, which will process their service request. We denote the edge cloud covering user u by e_u. Additionally, we denote the set of users an edge cloud e covers by $\mathcal{U}_e = \{u \in \mathcal{U} : e_u \equiv e\}$. The service some user u requests is denoted by s_u. When submitting a request, users also provide thresholds for accuracy and delay to inform the MEC system for how to make decisions with respect to QoS. The accuracy threshold provided by user u is denoted by $\alpha_u \in [0,1]$; the delay threshold provided by user u is denoted by $\delta_u \in [0, \delta_{\max}]$, where $\delta_{\max}$ represents the maximum possible delay. These thresholds are used to prioritize the needs of end users. More specifically, some applications that use deep learning are more sensitive to inaccurate answers and others are more time-sensitive. For instance, a self-driving vehicle that uses object detection to detect nearby pedestrians would be more accuracy-sensitive and delay-sensitive than a game on a smartphone.

Service Models. We consider a set of services that are available for users to request, denoted by $\mathcal{S} = \{1, \cdots, S\}$ where S is the number of available services. We assume that there is at least one implementation for each service $s \in \mathcal{S}$. However, we also allow for EI services to be implemented by several different machine learning architectures. For brevity, we refer to a single service implementation as a "service model" for short. The set of implemented service models for service s is denoted by $\mathcal{M}_s = \{1, \cdots, m_s\}$ where $m_s \geq 1$ is the number of models implemented for service s. For simplicity, we also denote the set of all individual implemented service models by $\mathcal{SM} = \{(s,m) : \forall s \in \mathcal{S}, m \in \mathcal{M}_s\}$. Each service model $(s,m) \in \mathcal{SM}$ is associated with an accuracy metric $A_{sm} \in [0,1]$. We assume this value is provided by evaluation using some test data-set (as is typical in machine learning). We denote the expected delay for performing service model (s,m) for user u by $D_{sm}(u)$ — this is defined more explicitly in §2.3.2. Finally, we denote communication, computation, and storage costs for each service model $(s,m) \in \mathcal{SM}$ by k_{sm}, w_{sm}, and r_{sm}, respectively.

2.3.2 Quality-of-Service (QoS) Definition

We consider QoS for EI models to be comprised of two components: provided model accuracy and incurred delay. As mentioned earlier in §2.3.1, each user request submitted to the system includes thresholds for requested minimum accuracy, α_u, and requested maximum delay, δ_u. As such, we use these threshold values to compute the expected QoS a service model (s,m) can provide to user u. The formal definition is provided below in Eq. (2.1),

$$Q(u,s,m) \triangleq \begin{cases} \frac{1}{2}[\hat{a}_{sm}(u) + \hat{d}_{sm}(u)] & \text{if } s = s_u \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

where $\hat{a}_{sm}(u)$ and $\hat{d}_{sm}(u)$ represent how much service model (s,m) satisfies user u's accuracy and delay thresholds, respectively. The summation of these two terms is multiplied by $1/2$ because the maximum possible values for both $\hat{a}_{sm}(\cdot)$ and $\hat{d}_{sm}(\cdot)$ is 1.0, thus normalizing the range to $Q(\cdot) \in [0,1]$. A key challenge when designing QoS functions for this work is addressing the fact that accuracy and delay naturally have different ranges. As such, it is necessary to ensure that these two metrics are made comparable to one another. The $\hat{a}_{sm}(\cdot)$ and $\hat{d}_{sm}(\cdot)$ functions are used for this reason. While experimenting with other QoS function definitions, it was common for delay to dominate because delay would often be much larger than a 1.0 (the maximum value for accuracy).

Accuracy Satisfaction

As stated, each user submits a minimum accuracy threshold, $\alpha_u \in [0,1]$, which indicates the amount of accuracy needed to satisfy them. This accuracy of a service model, A_{sm}, is a metric retrieved from model evaluation (as is standard with machine learning models). We define this as a nonlinear function in Eq. (2.2) below,

$$\hat{a}_{sm}(u) = \begin{cases} 1 & \text{if } A_{sm} \geq \alpha_u \\ \max(0, 1 - (\alpha_u - A_{sm})) & \text{otherwise} \end{cases} \tag{2.2}$$

where the first case represents when a user's accuracy request has been met and the second case provides reduced satisfaction based on the difference between user-requested accuracy and the evaluated accuracy for service model (s,m).

Delay Satisfaction

Similarly, each user submits a maximum delay threshold, $\delta_u \in [0, \delta_{\max}]$, to indicate the amount of accuracy they are willing to tolerate. If they receive a response for their request within $\delta_{\max}$ time units, then they are satisfied; otherwise, their satisfaction will degrade. The formal definition is provided below in Eq. (2.3),

$$\hat{d}_{sm}(u) = \begin{cases} 1 & \text{if } D_{sm}(u) \leq \delta_u \\ \max\left(0, 1 - \frac{D_{sm}(u) - \delta_u}{\delta_{\max}}\right) & \text{otherwise} \end{cases} \tag{2.3}$$

where $D_{sm}(u)$ is the expected delay from processing user u's request using service model (s,m). This is defined below in Eq. (2.4) as the sum of two terms of the transmission delay, $D^{tran}_{sm}(\cdot)$, and the computation delay, $D^{comp}_{sm}(\cdot)$. See below for the formal definition:

$$D_{sm}(u) = D^{tran}_{sm}(u) + D^{comp}_{sm}(u). \tag{2.4}$$

The transmission delay is a function of the communication cost of service model (s,m) and the communication capacity of user u's covering edge cloud, e_u. Additionally, the edge cloud's bandwidth is evenly shared across all of the users it covers — see Eq. (2.5),

$$D^{tran}_{sm}(u) = \frac{k_{sm}}{K_{e_u}/|\mathcal{U}_{e_u}|} = \frac{k_{sm}\,|\mathcal{U}_{e_u}|}{K_{e_u}} \tag{2.5}$$

where k_{sm} is the communication cost for service model (s,m). Similarly, computation delay is a function of the computation cost of service model (s,m) and user u's covering edge cloud, e_u, and its computation resources — see Eq. (2.6),

$$D^{comp}_{sm}(u) = \frac{w_{sm}}{W_{e_u}/|\mathcal{U}_{e_u}|} = \frac{w_{sm}\,|\mathcal{U}_{e_u}|}{W_{e_u}} \tag{2.6}$$

where w_{sm} is the computation cost for service model (s,m). We assume that the an edge cloud's computation capacity is evenly shared across its covered users.

2.4 Problem Definition

Given the system model, we now define the *Placement for Intelligent Edge Services* (PIES) problem. PIES performs service placement for EI models with multiple implementations with the goal of maximizing QoS, defined in Eq. (2.1). On top of making placement decisions, the PIES problem also decides *which* placed service model will serve a user's request if there are more implementations for a requested service available. For instance, given each user u makes a request for service s_u, if u's covering edge cloud e_u has had more than 1 model of service s_u placed on it, then the PIES problem will also select a placed model to process u's request.

2.4.1 PIES Formulation

The PIES problem is defined as an *integer linear program* (ILP) and consists of two types of decisions: *(i)* model placement and *(ii)* model scheduling. For the former, we consider a binary decision variable $\mathbf{x} = (x_e^{sm})_{\forall e \in \mathcal{E}, s \in \mathcal{S}, m \in \mathcal{M}_s} = 1$ if service model (s,m) is placed on edge cloud e, 0 otherwise. For the latter, we consider another binary decision variable $\mathbf{y} = (y_u^m)_{\forall u \in \mathcal{U}, m \in \mathcal{M}_{s_u}} = 1$ if user u's service request is served by its covering edge cloud e_u with service model (s_u,m), 0 otherwise. We formally define the PIES problem below:

$$\text{maximize} \sum_{u \in \mathcal{U}} \sum_{m \in \mathcal{M}_{s_u}} y_u^m Q(u, s_u, m) \tag{2.7}$$

$$\text{subject to} \sum_{m \in \mathcal{M}_{s_u}} y_u^m \leq 1 \qquad \forall u \in \mathcal{U} \tag{2.7a}$$

$$\sum_{s \in \mathcal{S}} \sum_{m \in \mathcal{M}_s} x_e^{sm} r_{sm} \leq R_e \qquad \forall e \in \mathcal{E} \tag{2.7b}$$

$$y_u^m \leq x_{e_u}^{s_u m} \qquad \forall u \in \mathcal{U}, m \in \mathcal{M}_{s_u} \tag{2.7c}$$

$$x_e^{sm} \in \{0,1\} \qquad \forall e \in \mathcal{E}, (s,m) \in \mathcal{SM} \tag{2.7d}$$

$$y_u^m \in \{0,1\} \qquad \forall u \in \mathcal{U}, m \in \mathcal{M}_{s_u} \tag{2.7e}$$

The objective function is defined in Eq. (2.7) and maximizes the expected QoS provided by the edge clouds to all users. Constraint (2.7a) ensures that no more than 1 model is used to process a user's request. Constraint (2.7b) guarantees that all edge clouds' storage capacities are not exceeded by the summation of the storage costs of their placed service models. Constraint (2.7c) ensures that users can only be served if their covering edge cloud has placed at least 1 implementation of their requested service. Finally, constraints (2.7d) and (2.7e) defines $\mathbf{x}$ and $\mathbf{y}$ as binary decision variables.

2.4.2 PIES Problem Complexity & Properties

Next, we provide proofs related to the hardness of the PIES problem, as well as theoretical properties that can be used to provide approximation guarantees for algorithm design.

Theorem 1. *The service model placement sub-problem of the PIES problem is NP-hard.*

Proof. We prove Theorem 1 using a reduction from the 0/1 Knapsack problem, which is one of Karp's 21 classical problems proven to be NP-*complete* [57, 58, 59]. To review, the 0/1 Knapsack

problem considers $1, \cdots, n$ items with each item having a weight cost c_i and a value v_i, as well as a maximum capacity C for the knapsack. The problem's objective is to maximize $\sum_{i=1}^{n} v_i x_i$ subject to $\sum_{i=1}^{n} w_i x_i \leq C$ and $x_i \in \{0,1\}$. Here, $x_i = 1$ if and only if the i^{th} item is selected to be placed in the knapsack.

We can reduce the 0/1 Knapsack problem to the PIES problem as follows. Suppose that there are $|\mathcal{S}| = n$ available services in the MEC system and only 1 model per service, i.e., $|\mathcal{M}_s| = 1$ $(\forall s \in \mathcal{S})$. Each service i has v_i users requesting it — meaning there are $|\mathcal{U}| = \sum_{i=1}^{n} v_i$ users in total. Suppose there is $|\mathcal{E}| = 1$ edge cloud in the MEC system with storage capacity $R_1 = C$, communication capacity $K_1 = \infty$, and computation capacity $W_1 = \infty$. Let all users be covered by this 1 edge cloud, such that $e_u = 1$ $(\forall u \in \mathcal{U})$. Let the storage costs associated with each service model be $r_{s1} = c_s$ $(1 \leq s \leq n)$ and the communication/computation costs $k_{s1} = 1, w_{s1} = 1$ $(1 \leq s \leq n)$. We assume that the QoS requirements of all users are relaxed, meaning that $\alpha_u = 0.0, \delta_u = \delta_{\max}$ $(\forall u \in \mathcal{U})$. Then we claim that the 0/1 Knapsack problem is feasible if and only if we can maximize expected QoS across all users, i.e., the optimal decision variable of the constructed PIES instance equals $x_1^{i1} \equiv x_i$. First, given the optimal solution to the 0/1 Knapsack, placing the services corresponding to the decisions in x_i on the single edge cloud in our constructed instance gives the optimal solution to the PIES problem that maximizes QoS by maximizing the number of user requests served on the edge. Moreover, given an optimal solution to the PIES problem that maximizes QoS across all users, placing the corresponding items in the knapsack gives an optimal solution to the 0/1 Knapsack problem. Given the decision problem of the 0/1 Knapsack problem is NP-*complete*, it follows that the PIES problem is NP-*hard*. This concludes the proof. □

Theorem 2. *Given a service model placement, $\boldsymbol{x}$, the optimal solution to the PIES model scheduling sub-problem is given by a greedy algorithm.*

To prove Theorem 2, we consider an auxiliary, undirected multigraph and discuss its construction. Consider a multigraph $\mathcal{G} = (V, L, f)$ where V is the set of nodes, $L \subseteq |V| \times |V|$ the set of links/edges (hereafter we will refer to as *links*), and $f : L \to \{\{u, v\} : u, v \in V\}$ is the link identifier function. Consider the node set V to be composed of 3 sets of nodes: a set containing a single *root node* V_R, a set of *user nodes* V_U, and a set of *service nodes* V_S. Thus, we say $V = V_R \cup V_U \cup V_S$. We define V_U as the set of users with at least 1 model placed on their edge cloud for their requested service, i.e., $V_U = \{u \in \mathcal{U} : \sum_{m \in M_{s_u}} x_{e_u}^{s_u m} > 0\}$. We define V_S as the set of requested services for each user $u \in V_U$ that have had at least 1 service model placed on their covering edge cloud, i.e.,

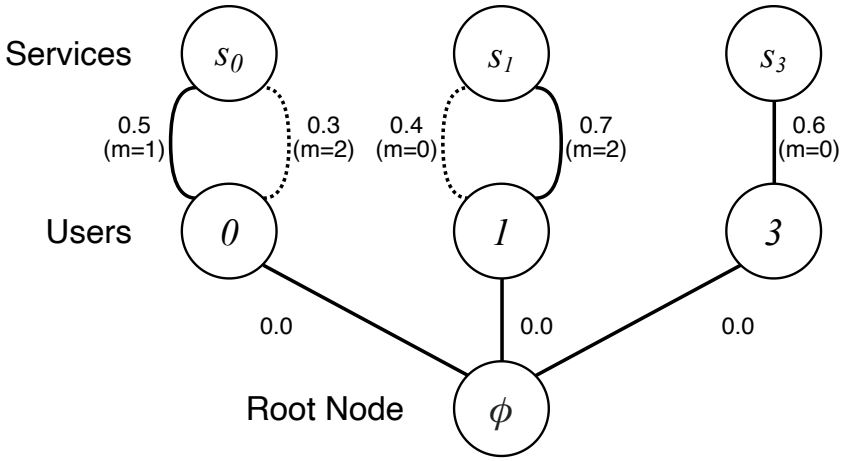

Figure 2.2: Example auxiliary multigraph representing PIES scheduling sub-problem. Links between user/service nodes correspond to an model implementation for a user's requested service placed on the user's covering edge. Float values correspond with QoS weights. Solid links compose the maximum spanning tree for optimal model scheduling.

$V_S = \{s_u : \forall u \in V_U\}$. We note that we allow for duplicate services such that each user $u \in V_U$ has its own service node. The set of links, L, contains links between node sets V_R, V_U and V_U, V_S. First, there is 1 edge between the root node, $\phi \in V_R$, and each of the user nodes $u \in V_U$, i.e., $\{(\phi, u) : \forall u \in V_U\} \subseteq L$. Then, a user node $u \in V_U$ can have multiple links to its corresponding service node $s_u \in V_S$. The number of links between $u \in V_U$ and $s_u \in V_S$ is equal to $\sum_{m \in \mathcal{M}_{s_u}} x_{e_u}^{s_u m}$, which is the number of model implementations for s_u placed on u's covering edge cloud, e_u. Finally, each link is weighted. Links between the root node $\phi \in V_R$ and user nodes $u \in V_U$ are weighted by 0. Links between each user nodes $u \in V_U$ and its requested service nodes s_u are weighted by the expected $Q(u, s, m)$ where m corresponds with the model type of the respective link between u and s_u. A simple example of a multigraph constructed by this method is provided in Fig. 2.2.

Proof. Given a service placement decision, $\mathbf{x}$, we can prove Theorem 2 by constructing an auxiliary multigraph that represents the problem space for the PIES model scheduling sub-problem as described above and shown in Fig. 2.2. With this graph constructed, it is easy to verify that the optimal solution to the model scheduling sub-problem is equivalent to finding a maximum spanning tree (MST) — which can be found by applying a greedy algorithm for minimum spanning trees (e.g., Kruskal's algorithm [60]) and negating all of the edge weights. This concludes the proof. □

Proving Submodularity. In order to provide an approximation guarantee for the NP-*hard* PIES placement sub-problem, we prove the PIES service placement sub-problem is maximizing a monotone submodular set function, we rewrite our problem as a set optimization. Let $P(\mathbf{x}) \triangleq \{(e, (s, m)) \in$

$\mathcal{E} \times \mathcal{SM} : x_e^{sm} = 1\}$ denote the set of service model placements according to decision variable $\mathbf{x}$, where $(e,(s,m))$ means service model (s,m) is placed on edge cloud e. Next, let $\sigma(P(\mathbf{x}))$ denote the optimal objective value of PIES for a given $\mathbf{x}$. By writing $P(\mathbf{x})$ as P, we can rewrite the PIES placement sub-problem as:

$$\max \ \sigma(P) \tag{2.8}$$

$$\text{s.t.} \sum_{(e,(s,m))\in P\cap P_e} r_{sm} \leq R_e \qquad \forall e \in \mathcal{E} \tag{2.8a}$$

$$P \subseteq \mathcal{E} \times \mathcal{SM} \tag{2.8b}$$

where $P_e \triangleq \{e\} \times \{(s,m) \in \mathcal{SM} : r_{sm} \leq R_e\}$ is the set of all possible *single* service model placements at edge cloud e. From here, we observe the following:

- *Matroid constraint:* Let $\mathcal{I}$ be the collection of all P satisfying constraints (2.8a), (2.8b). It is then easy to verify that $\mathbb{M} = (\mathcal{E} \times \mathcal{SM}, \mathcal{I})$ is a matroid. This is known as the *partition matroid* as $\{P_e\}_{e\in\mathcal{E}}$ is a partition of the ground set $(\mathcal{E} \times \mathcal{SM})$.
- *Monotone submodular objective function:* We show that the objective function (2.8) has the following properties.

Theorem 3. *Function $\sigma(P)$ is monotone increasing and submodular for any feasible P.*

Proof. First, adding an element $(e,(s,m))$ to P corresponds to adding new possible users to serve or new links in the auxiliary multigraph $\mathcal{G}$ (see Fig. 2.2). Under either outcome, the objective value, Eq. (2.7), will either increase or remain unchanged. Thus it is sufficient to say that $\sigma(P)$ is monotone increasing.

The PIES service model placement sub-problem is submodular iff for every $A, B \subseteq \mathcal{E} \times \mathcal{SM}$ where $A \subseteq B$ and some $p \notin B$, the following condition holds: $\sigma(A \cup \{p\}) - \sigma(A) \geq \sigma(B \cup \{p\}) - \sigma(B)$. We can define the objective value under optimal scheduling given a set of placements P as

$$\sigma(P) \triangleq \sum_{u\in\mathcal{U}} \sigma_u(P) \tag{2.9}$$

where $\sigma_u(P)$ is the optimal QoS provided to user u with service model placements P. We can define $\sigma_u(P)$ as follows:

$$\sigma_u(P) \triangleq \max_{(e,(s,m))\in P} \{Q(u,s,m) : e = e_u \wedge s = s_u\} \cup \{0\}. \tag{2.10}$$

Next, we claim that for every user $u \in \mathcal{U}$ the following holds: $\sigma_u(A \cup \{p\}) - \sigma_u(A) \geq \sigma_u(B \cup \{p\}) - \sigma_u(B)$. We can verify this by elaborating on what these expressions represent and by exploiting their definitions. First, by definition, $\sigma_u(A \cup \{p\}) - \sigma_u(A)$ represents the increase $\{p\}$ provides to the objective. It should be noted that $\{p\}$ can only provide increase if its provided QoS for u is greater than what A could already provide. Additionally, we note $\sigma_u(A \cup \{p\}) \in \{\sigma_u(A), \sigma_u(\{p\})\}$. We claim this because $\sigma_u(A \cup \{p\}) \equiv \sigma_u(A)$ if $\sigma_u(A) \geq \sigma_u(\{p\})$, meaning an already placed model in A is scheduled to serve u's request, or $\sigma_u(A \cup \{p\}) \equiv \sigma_u(\{p\})$ if $\sigma_u(A) < \sigma_u(\{p\})$, meaning the new service model placed by the new placement $\{p\}$ is scheduled to serve u's request. These observations similarly hold for B as well. Now, we prove the following inequality $\sigma_u(A \cup \{p\}) - \sigma_u(A) \geq \sigma_u(B \cup \{p\}) - \sigma_u(B)$ directly by its possible cases:[1]

- **Case 1.** $\sigma_u(A \cup \{p\}) \equiv \sigma_u(A)$ and $\sigma_u(B \cup \{p\}) \equiv \sigma_u(B)$. It is easy to verify that $\sigma_u(A \cup \{p\}) - \sigma_u(A) \geq \sigma_u(B \cup \{p\}) - \sigma_u(B)$ becomes $0 \geq 0$, which holds.

- **Case 2.** $\sigma_u(A \cup \{p\}) > \sigma_u(A)$ and $\sigma_u(B \cup \{p\}) \equiv \sigma_u(B)$. It is easy to verify that the lefthand side of the original inequality becomes a value > 0 and the righthand side becomes 0, thus the inequality holds.

- **Case 3.** $\sigma_u(A \cup \{p\}) > \sigma_u(A)$ and $\sigma_u(B \cup \{p\}) > \sigma_u(B)$. Since $A \subseteq B$, any service model placed under A is also placed under B. Intuitively, any increase to QoS for user u provided by $A \cup \{p\}$ can be matched by $B \cup \{p\}$ because B has every service model to serve u's request that A has. Additionally, B could also have service models that provide greater QoS for u than A due to it having more service models placed. Thus, it must follow that the original inequality holds because A's increase in QoS for u is always at least as large as B's increase.

Thus, $\sigma_u(\cdot)$ is submodular for every $u \in \mathcal{U}$. Since any function that is a summation of submodular functions is also submodular [61], it then follows that $\sigma(\cdot)$ is also submodular. This concludes the proof. □

2.5 Efficient Algorithm Design

[1] Note the case that $\sigma_u(A \cup \{p\}) \equiv \sigma_u(A)$ and $\sigma_u(B \cup \{p\}) > \sigma_u(B)$ can never occur. This is due to the fact that $A \subseteq B$ and thus if $\{p\}$ provides greater QoS for user u than B, then it follows that is also true for A.

Algorithm 1: Optimal Model Scheduling (OMS)

Input : Service placement ($\mathbf{x}$), Input parameters of (2.7)
Output: Model scheduling ($\mathbf{y}$)

Initialize $\mathbf{y} \leftarrow (y_u^m = 0)_{\forall u \in \mathcal{U}, m \in \mathcal{M}}$;
foreach *user* $u \in \mathcal{U}$ **do**
    if $\sum_{m \in \mathcal{M}_{s_u}} x_{e_u}^{s_u m} > 0$ **then**
        $m^* \leftarrow \arg\max_{m \in \mathcal{M}_{s_u}} \{Q(u, s_u, m), \forall u \in \mathcal{U}_e : x_{e_u}^{s_u m} = 1\}$;
        $y_u^{m^*} \leftarrow 1$;
return $\mathbf{y}$;

2.5.1 Scheduling Sub-Problem

In §2.4.2, we proved the PIES model scheduling sub-problem can be optimally solved with a greedy solution (see Theorem 2). As such, we introduce a simple greedy algorithm, *Optimal Model Scheduling* (OMS). The pseudocode is provided in Algorithm 1. OMS works in a straightforward fashion. Given a service placement decision and the PIES input parameters, OMS iterates through each user $u \in \mathcal{U}$ and if there is at least one model of their requested service, s_u, on their covering edge cloud, e_u, then OMS selects the model that provides the greatest QoS to user u. The runtime for OMS is $O(|\mathcal{U}||\mathcal{M}_s^{\max}|)$ where $|\mathcal{M}_s^{\max}| = \max_{s \in \mathcal{S}}(|M_s|)$.

2.5.2 Placement Sub-Problem

Here, we introduce two algorithms that can be used to solve the service model placement sub-problem for PIES. The first is an approximation algorithm that exploits the theoretical properties of the PIES objective (discussed in §2.4.2) to achieve an approximately optimal solution. The latter algorithm mimics some of the logic of this algorithm while reducing computational heft.

Approximation Algorithm

Because the PIES problem aims to maximize a monotone increasing, submodular set function under matroid constraints (see Theorem 3), a standard greedy algorithm can provide a $(1 - 1/e)$-approximation of the optimal solution [62]. As such, we introduce *Approximate Greedy Placement* (AGP) which serves as an approximation algorithm for the PIES placement sub-problem. Its pseudocode is provided in Algorithm 2. AGP iterates through each edge cloud $e \in \mathcal{E}$ and, in each iteration, it finds the set of service models $(s, m) \in \mathcal{SM}$ that can be placed on e without violating the storage capacity constraint. It then computes the objective value using optimal model schedul-

Algorithm 2: Approx. Greedy Placement (AGP)

Input : Input parameters of (2.7)
Output: Service placement ($\mathbf{x}$)

1 Initialize $\mathbf{x} \leftarrow (x_e^{sm} = 0)_{\forall e \in \mathcal{E}, s \in \mathcal{S}, m \in \mathcal{M}}$;
2 $P \leftarrow \{\}$; // Placement decisions $\forall e \in \mathcal{E}$.
3 **foreach** $e \in \mathcal{E}$ **do**
4 $\quad \hat{P} \leftarrow \{\}$; // Placement decisions for this e.
5 $\quad \hat{R} \leftarrow R_e$;
6 $\quad$ **repeat**
7 $\quad\quad L \leftarrow \{(s,m) \in \mathcal{SM} \setminus \hat{P} : r_{sm} \leq \hat{R}\}$;
8 $\quad\quad s^*, m^* \leftarrow \underset{(s,m) \in L}{\arg\max}\, \sigma(P \cup \{(e,(s,m))\})$;
9 $\quad\quad \hat{P} \leftarrow (s^*, m^*)$;
10 $\quad\quad P \leftarrow P \cup \{(e,(s^*,m^*))\}$;
11 $\quad\quad \hat{R} \leftarrow \hat{R} - r_{s^*m^*}$;
12 $\quad$ **until** $|L \setminus \hat{P}| = 0$;
13 **foreach** $(e,(s,m)) \in P$ **do**
14 $\quad x_e^{sm} \leftarrow 1$;
15 **return** $\mathbf{x}$;

ing via Eq. (2.9) to find the immediate best choice. Once there are no more legitimate choices to choose from, it moves on to the next edge cloud. Once iteration through edge clouds is finished, it converts the placement decisions (represented as a set) into the standard format for the decision variable. However, AGP's runtime is not desirable due to its need to compute optimal scheduling for each possible option at each iteration.

Efficient Algorithm

Due to the heavy runtime complexity of AGP, there is a need for a more efficient algorithm that can relatively match AGP's performance with respect to approximating the optimal solution without the large computational cost. Thus, we introduce the *Efficient Greedy Placement* (EGP) algorithm. EGP iteratively places models by keeping a record of anticipated benefit of placing any given service model on an edge cloud. It does this without computing optimal scheduling, thus reducing its computational cost. EGP's pseudocode is provided in Algorithm 3. In line 1, the placement decision variable is initialized. Then, on line 2, we begin to iterate through each edge cloud to decide which service models should be placed on the current edge cloud. Line 3 initializes an empty hash-map and lines 4-6 compute the total QoS each service model relevant for the current edge cloud can provide towards the objective function. This data structure will be updated as decisions are made. Lines 7-9 initialize some supporting variables for EGP's logic. $\mathcal{A}$ records service models that have been considered for placement at some point for the current edge cloud; $\mathcal{B}$ keeps track of the set

Algorithm 3: Efficient Greedy Placement (EGP)

Input : Input parameters of (2.7)
Output: Service placement ($\mathbf{x}$)

1 Initialize $\mathbf{x} \leftarrow (x_e^{sm} = 0)_{\forall e \in \mathcal{E}, s \in \mathcal{S}, m \in \mathcal{M}}$;
2 **foreach** *edge* $e \in \mathcal{E}$ **do**
3 Initialize hash-map $\mathbf{v}$ with default values of 0.0;
4 **foreach** *user* $u \in \mathcal{U}_e$ **do**
5 **foreach** *model type* $m \in \mathcal{M}_{s_u}$ **do**
6 $\mathbf{v}_{s_u m} \leftarrow \mathbf{v}_{s_u m} + Q(u, s_u, m)$;
7 $\mathcal{A} \leftarrow \{\}$; // Considered service models.
8 $\mathcal{B} \leftarrow \{\}$; // Satisfied users.
9 $\hat{R} \leftarrow R_e$; // Remaining storage.
10 **repeat**
11 $s^*, m^* \leftarrow \underset{(s,m) \in \text{keys}(\mathbf{v}) \setminus \mathcal{A}}{\arg\max} \{\mathbf{v}_{sm}\}$;
12 **if** $r_{s^* m^*} \leq \hat{R}$ **then**
13 $x_e^{s^* m^*} \leftarrow 1$;
14 $\hat{R} \leftarrow \hat{R} - r_{s^* m^*}$;
15 **foreach** $m \in \mathcal{M}_{s^*}$ *where* $(s^*, m) \notin \mathcal{A}$ **do**
16 $\mathbf{v}_{sm} \leftarrow \sum_{u \in \mathcal{U}_e \setminus \mathcal{B}} Q(u, s^*, m) - Q(u, s^*, m^*)$;
17 $\mathcal{A} \leftarrow \mathcal{A} \cup \{(s^*, m^*)\}$;
18 **foreach** $u \in \mathcal{U}_e$ **do**
19 **if** $Q(u, s^*, m^*) = 1$ **then** $\mathcal{B} \leftarrow \mathcal{B} \cup \{u\}$;
20 **until** $(\hat{R} = 0) \vee (|\mathcal{U}_e| = |\mathcal{B}|) \vee (|\mathcal{A}| = |keys(\mathbf{v})|)$;
21 **return** $\mathbf{x}$;

of users who can be provided maximum QoS; and $\hat{R}$ tracks the remaining storage capacity. Lines 10-20 find the service model (s^*, m^*) that provide the maximum QoS among the service models we have yet to consider from our hash-map. If (s^*, m^*) can be placed without violating the storage constraint, then it will be placed (line 13) and remaining storage will be reduced (line 14). Then, in lines 15-16 we recalculate the benefit of each other model implementation for s^* by computing $\sum_{u \in \mathcal{U}_e \setminus \mathcal{B}} Q(u, s^* m) - Q(u, s^*, m^*)$. Since the newly placed model degrades the benefit from picking other implementations of the same service, we sum the difference between these other models and the newly placed model to reevaluate the benefit of placing them. Lines 10-20 repeat until either there is no more storage space, all of the edge cloud's user's achieve maximum QoS, or we have considered all models relevant for the edge cloud (according to the keys recorded in the hash-map). The runtime complexity of EGP is $O(|\mathcal{U}| + |\mathcal{U}||\mathcal{M}_s^{\max}|)$ where $|\mathcal{M}_s^{\max}| = \max_{s \in \mathcal{S}}(|M_s|)$.

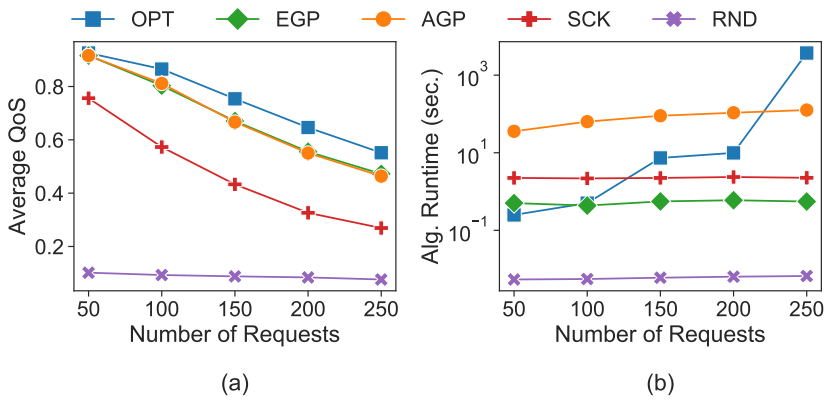

Figure 2.3: **Validation Test.** For these experiments, we consider $|\mathcal{U}| = [50, 100, 150, 200, 250]$, with 10 trials each. We perform this validation test to confirm the efficacy of EGP relative to the optimal solution and the approximation algorithm, AGP.

2.6 Experimental Design & Results

We consider numerical simulations and a real-world implementation with real image data and ML models. Algorithms are implemented in Python 3.8 and experiments are largely run on a macOS 64 bit machine with a 3.2 GHz quad-core Intel Core i5 processor and 32 GB 1600 MHz DDR3 memory.

2.6.1 Baselines

We compare AGP and EGP to the ILP defined in Eq. (2.7) which is solved using the PuLP Python library [63] and the CBC solver [64] (referred to as "OPT"). We also adapt the standard dynamic programming algorithm for the 0/1 Knapsack problem for the PIES problem (referred to as "SCK"). SCK considers the individual service models as the separate items — with their storage costs serving as their weights and Eq. (2.1) as their values. SCK will use Alg. 1 for scheduling. We also consider a random placement and scheduling heuristic (referred to as "RND") as our other baseline.

2.6.2 Numerical Simulations

We sample uniformly random integer values for edge capacities using the following distributions $K_e, W_e \in [300, 600]$, and $R_e \in [100, 200]$ $(\forall e \in \mathcal{E})$. Service model storage costs are similarly uniformly sampled integer values where $k_{sm}, w_{sm} \in [15, 30]$ and $r_{sm} \in [10, 20]$ $(\forall (s,m) \in \mathcal{SM})$. Service models' cached accuracy, A_{sm}, are sampled from a Gaussian distribution with a mean of 0.65 and a

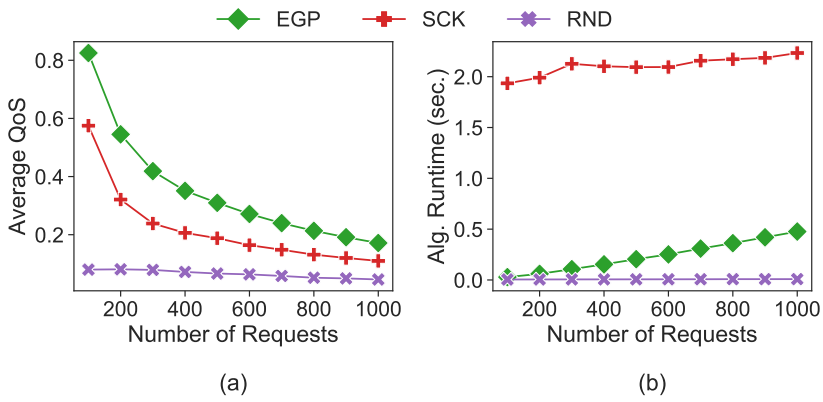

Figure 2.4: **Numerical Results.** Experiments with more user requests $|\mathcal{U}| = [100, 200, \cdots, 1000]$, with 100 trials each.

standard deviation of 0.1 (sampled values are clipped to the range $[0,1]$). We assume users request service types from $\mathcal{S}$ uniformly at random. User accuracy thresholds, α_u, are set to $1-\varepsilon$ where ε is sampled from an exponential distribution clipped to the range $[0,1]$ with $\lambda = 0.125$. User delay thresholds, δ_u, are set to a sampled value (clipped to range $[0, \delta_{\max}]$ where $\delta_{\max} = 10$ seconds) from an exponential distribution where $\lambda = 1.5$. Finally, we consider $|\mathcal{E}| = 10$ edge clouds, $|\mathcal{S}| = 100$ services with each service having a random number of implementations in the range of $[1,10]$ (sampled uniformly). We increase the number of users for experiments.

First, we compare our proposed algorithms (EGP and AGP) to the optimal solution. Due to the hardness of the PIES problem, we consider a validation case to demonstrate EGP performance relative to the optimal solution[2] and AGP. In Fig. 2.3a, we see that the AGP and EGP algorithms perform well for approximating the optimal solution provided by the solver. More importantly, we find that EGP is able to match AGP's performance even though it has a proven approximation bound. Specifically, we find that, on average, AGP and EGP achieve an approximation ratio of 0.900 and 0.904, respectively.[3] In Fig. 2.3b, we see that EGP greatly outclasses both Optimal and AGP in terms of efficiency. The excessive cost associated with AGP's runtime is due to its reliance on performing optimal model scheduling for each candidate service model at each selection step. EGP also manages to best SCK. When considering more requests, we see in Fig. 2.4 that our EGP solution achieves roughly 50% more QoS than SCK while still managing to be more efficient.

[2]It is worth noting in some larger scenarios we ran, it took over 20 hours for the optimal solver to complete.

[3]These values are computed by taking the QoS for each algorithm in one experiment trial and dividing it by the QoS provided by the optimal solution.

2.6.3 Real-World Implementation

We consider a simple real-world setup using set of 2 Nvidia Jetson Nano and 1 Raspberry Pi 3B+ nodes as IoT devices and a 2013 Apple iMac serving as the edge cloud. Each IoT device hosts roughly a third of the 2012 ImageNet dataset [65] via non-overlapping subsets. Each IoT device submits 100 requests with randomly sampled images from these data. Requests are submitted wirelessly for image classification service models hosted on the edge cloud. Here we focus on the multi-implementation aspect of the PIES placement sub-problem where multiple implementations for image classification can be placed and how this affects the QoS in the real-world case. Using PyTorch [66], we evaluate pre-trained image classification models to record their accuracy metric over the ImageNet 2012 data and record the average time needed for each model to perform inference, see Table 2.1. The edge cloud can place $|R_e| = 1$ model where each ML model is associated with $r_{sm} = 1$ storage cost. Since all the models accept the same data size, we fix the communication costs $w_{sm} = 1$ for all the ML models. The communication and computation capacities for the edge cloud are robustly tuned to match the real-world computation and communication delay. Similarly to the numerical simulation setup, each request's α_u is sampled from $1 - \varepsilon$ where ε is sampled from an exponential distribution clipped to the range $[0.0, 1.0]$ with a rate parameter $\lambda = 0.0625$. Each sample's delay threshold δ_u is sampled from a Gaussian distribution with a mean of 0.5 and a standard deviation of 0.125, clipped to the range $[0, \delta_{\max}]$ where $\delta_{\max} = 1.0$ second. The QoS for each request is calculated using Eq. (2.1) using the real-time incurred latency (in seconds) and the evaluated model accuracy.

In Fig. 2.5a, we see that all considered algorithms but random are able to match the optimal solution — with all non-random algorithms exclusively placing MobileNet in Fig. 2.5b. In Fig. 2.5a, the QoS distribution of the non-random algorithms are much more concentrated on the upper end when compared to random. In this setup, random does better than in Figs. 2.3, 2.4 because a request will never be dropped (i.e., there is always an image classification available to provide *some* QoS). Thus, future real-world implementations must consider various service types. For the meantime, these results show promise but could be improved and expanded upon by considering a more robust real-world setup with more EI service types (e.g., speech-to-text, video classification).

Table 2.1: Image classifications models used for the real-world implementation with model accuracy metrics and average computational delay from evaluation with ImageNet 2012 data.

Models	Accuracy, A_{sm}	Avg. Comp. Delay (sec.)
AlexNet [67]	56.52%	0.04
DenseNet [68]	77.14%	0.47
GoogLeNet [69]	69.78%	0.13
MobileNet [70]	71.88%	0.06
ResNet [71]	69.76%	0.08
SqueezeNet [72]	58.09%	0.07

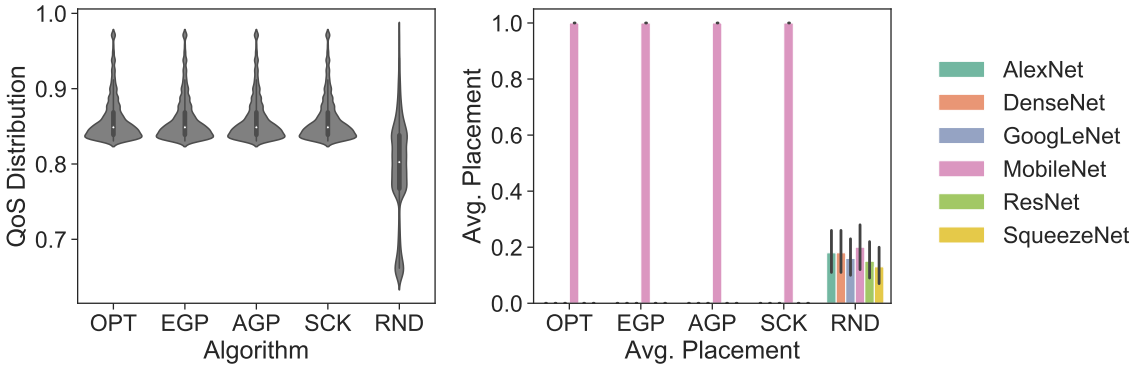

Figure 2.5: **Real world implementation.** (5a) QoS distribution achieved by each of the algorithms. (5b) Average placement decision for each image classification model across 100 trials.

2.7 Conclusions

The PIES problem, to the best of our knowledge, is the first service placement and scheduling problem that explicitly considers the case of EI services having multiple implementations available for the same service. We proved that the PIES problem is NP-*hard* and prove a greedy set optimization algorithm can provide a $(1-1/e)$-approximation guarantee of the optimal solution. We then introduce a streamlined greedy algorithm that empirically matches this algorithm's performance with much greater efficiency. While these results are preliminary, they serve as a foundational first step towards this breed of service placement. For future work, we plan to consider more dynamic extension of this work where service placement decisions are made over a time horizon rather than all at once. Additionally, we will expand the real-world setup to include more EI services (e.g., video classification) with multiple implementations for placement and scheduling.

3

Joint Compression and Offloading Decisions for Deep Learning Services in 3-Tier Edge Systems

©2021 IEEE. Reprinted with permission from Minoo Hosseinzadeh and Nathaniel Hudson, "Joint Compression and Offloading Decisions for Deep Learning Services in 3-Tier Edge Systems," 2021 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN), December 2021, DOI: 10.1109/DySPAN53946.2021.9677398

Task offloading in edge computing infrastructure remains a challenge for dynamic and complex environments, such as Industrial Internet-of-Things. The hardware resource constraints of edge servers must be explicitly considered to ensure that system resources are not overloaded. Many works have studied task offloading while focusing primarily on ensuring system resilience. However, in the face of deep learning-based services, model performance with respect to loss/accuracy must also be considered. Deep learning services with different implementations may provide varying amounts of loss/accuracy while also being more complex to run inference on. That said, communication latency can be reduced to improve overall Quality-of-Service by employing compression techniques. However, such techniques can also have the side-effect of reducing the loss/accuracy provided by deep learning-based service. As such, this work studies a joint optimization problem for task offloading decisions in 3-tier edge computing platforms where decisions regarding task offloading are made in tandem with compression decisions. The objective is to optimally offload requests with compression such that the trade-off between latency-accuracy is not greatly jeopardized. We cast this problem as a mixed integer nonlinear program. Due to its nonlinear nature, we then decompose it into separate sub-problems for offloading and compression. An efficient algorithm is

proposed to solve the problem. Empirically, we show that our algorithm attains roughly a 0.958-approximation of the optimal solution provided by a block coordinate descent method for solving the two sub-problems back-to-back.

3.1 Introduction

Exponential technological growth has been a defining quality of the 21st century. This growth led to the development of the *Industrial Internet-of-Things* (IIoT) where smart sensors, devices, digital storage units, instruments, etc. are interconnected through the Internet to aid the needs of industry [73]. Industrial sectors (e.g., manufacturing [74]) can be supplemented by the IIoT by providing comprehensive data collection, sharing, and processing across network devices. A key enabling technology for IIoT is that of *Edge Computing* (EC) [33]. Under the EC paradigm, compute resources are pushed to the network edge to provide more opportunity for IIoT devices (e.g., smart sensors) to have data they collect/generate/sense processed more immediately than if they were to solely rely on a faraway central cloud server [75]. This is done by the deployment of edge servers (or edge servers) that process requests/data closer to end users. Following the growing popularity of EC, the notion of *Edge Intelligence* (EI) has also risen in popularity. EI pushes AI services, namely *Machine Learning* (ML) and *Deep Learning* (DL) services, to the network edge. EI allows for data collected/generated/sensed at the network edge to be immediately processed without needing to relay all IIoT-generated data to be sent to a faraway central cloud [76].

The benefits of EC [77] (e.g., reduced latency, increased scalability, improved reliability), making EI all the more attractive for providing reliable intelligent services [78, 79]. However, the hardware resources (e.g., communication, computation, and storage capacities) available at the edge servers are relatively constrained when compared to central cloud servers and these limitations must be explicitly considered [41, 80, 81]. Many works studying problems related to EC focus on optimizing the use of edge resources to best serve user requests One such problem is that of request offloading (or task offloading/scheduling). Request offloading to edge server-hosted services has been widely studied for both general [41, 82] and deep learning-based services [53, 54]. The problem of request offloading aims to decide where requests should be sent in the system to be processed (i.e., which edge server should serve a given request) while being mindful of system resources and incurred latency. Typically, the primary objective of offloading problems is to minimize latency experienced by end users [41, 46].

One viable approach for reducing the latency for transmitting data across communication channels is to reduce the size of data transmitted over the communication channel through compression techniques [83, 84, 85]. Beyond minimizing communication latency, compression has also been considered for the purposes of minimizing energy consumption incurred from data transmission in EC systems for general service requests [86]. However, the applicability of compression techniques faces an additional challenge for deep learning-based services. It is important to note that compression techniques can largely be split into 2 groups: *lossless* and *lossy*. Lossless compression does not compromise data quality when it is uncompressed; however lossy compression provides much greater data size reduction [87]. Since the performance of DL-based services relies on data quality, compression techniques that harm data quality can compromise the loss/accuracy of the DL-based service. Thus, there is a trade-off between how much we compress data in service requests to reduce latency while not jeopardizing model accuracy too greatly.

This chapter studies an offloading problem with the objective of optimizing the latency-accuracy trade-off for requests for DL-based services in 3-tier EC platforms while considering compression. We define this problem, ***C**ompression and **O**ffloading **D**ecisions on the **E**dge* (CODE), as a *mixed integer nonlinear program*. The CODE problem is similar to the problem studied in a prior work [53]. The novelty of the CODE problem is the joint consideration of both compression and offloading for DL-based service requests while considering more than 1 service request at a time. Further, CODE assumes edge servers can offload requests to other edge servers through device-to-device (D2D) communication channels. Finally, this chapter proposes a polynomial-time algorithm for approaching the CODE problem. Our algorithm is empirically shown to achieve a 0.958-approximation of the near-optimal solution which is given by a *block coordinate descent method* [88].

3.2 Related Work

Here, we briefly summarize the literature related to task offloading problems in edge computing systems. Prior works studying the request offloading problem in 3-tier EC platforms typically employ optimization techniques (e.g., linear programming, stochastic modeling) to either maximize the number of requests that were served under edge server hardware constraints [41, 80] or minimize overall latency provided to end users submitting requests [89, 90]. There are few recent works that jointly study offloading and compression decisions in EC systems. Xu et al. in [86] jointly studies resource allocation, task offloading, and data compression under edge server resource constraints

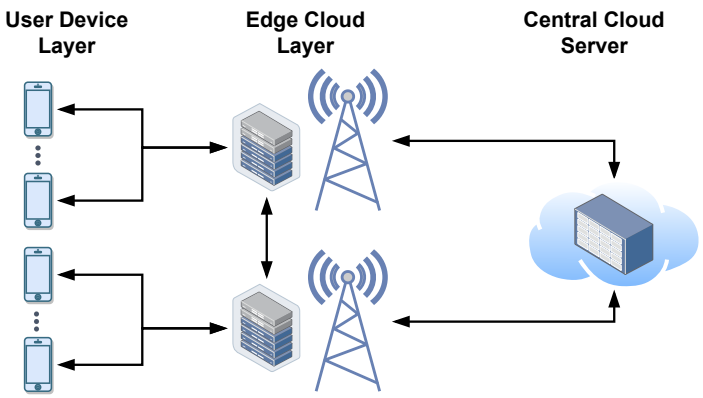

Figure 3.1: Illustrative overview of our considered 3-tier architecture.

while aiming to minimize energy consumption. They transform a non-convex optimization model into a convex problem and apply convex optimization to solve this joint problem. Elgendy et al. in [91] consider a similar joint optimization for resource allocation, data compression, and security. They show their non-convex joint optimization is NP-*hard* and use relaxation and linearization techniques to solve a convex version of the problem. However, a limitation of these prior works is that they are for general services available in EC systems. They do not consider the loss/accuracy provided by a DL-based service. Hosseinzadeh et al. in [54] study task offloading in a 3-tier EC platform while considering the latency-accuracy trade-off as the objective. However, that work fails to consider compression to reduce latency without greatly compromising model loss/accuracy. Zhao et al. in [53] study request offloading in 3-tier EC platforms by considering a simple environment where only 1 request is consider. Additionally, only 1 edge server is considered to simplify the problem.

The main contributions of our work to the task offloading problem include: *(i)* design of a mixed integer nonlinear program that jointly considers offloading and compression decisions for requests for DL-based services that aims to optimize the latency-accuracy trade-off; *(ii)* consideration of a complex 3-tier EC platform with multiple service requests and device-to-device (D2D) communication among edge servers to collaboratively process requests; and *(iii)* design of a polynomial-time iterative algorithm for solving CODE that empirically achieves a 0.958-approximation of the optimal solution provided by directly solving CODE using a *block coordinate descent method* [88].

3.3 System Design & CODE Problem Definition

3.3.1 3-Tier System Architecture

We consider a 3-tier edge computing system architecture. The topmost tier of this architecture is a central cloud server. The middle tier is a layer of edge servers deployed at the network edge. The bottom tier is the user device tier, comprised of IoT and IIoT devices. At any given time in the system, we consider a set of user requests submitted by the user devices in the user device layer. We denote the set of requests by $\mathcal{U}$. For this work, we assume each request $i \in \mathcal{U}$ corresponds with a single user (hence we may use "users" and "requests" interchangeably). We also denote the set of edge servers by $\mathcal{E}$. The edge server that directly covers a user device $i \in \mathcal{U}$ is denoted by w_i. The central cloud server will commonly be referred to by c throughout the chapter.

Before defining the problem, we clarify two points of the system. First, edge servers and cloud servers have a maximum number of requests they are able to process in the considered time window. We refer to this as the *admission capacity*. The admission capacity for edge server j is denoted by v_j^e and the admission capacity for the central cloud server is denoted by v^c. All entities in the system (i.e., user devices, edge servers, and the central cloud server) have hardware resources associated with them. Because this chapter is focused on offloading specifically, we only consider communication and computation resources. The notation for these will be presented later in §3.3.4.

3.3.2 Compression and Machine Learning Accuracy

In our prior work [53], we experimentally showed that image compression techniques based on DCT transforms, e.g., JPEG, reduce the expected accuracy of a machine learning model in a non-linear fashion when removing high frequency DCT coefficients, e.g., by giving them a zero value. More precisely, removing DCT coefficients initially has little to no effect in accuracy initially. There exists an inflexion point, dependent on the model, where additional compression results in dramatic loss of accuracy. We showed that this relationship between image compression and expected accuracy of a machine learning model can be fitted using the asymmetric Gompertz growth curve function [92, 93]:

$$g(s) = a \cdot e^{-b \cdot e^{-c \cdot s}} \tag{3.1}$$

where $a, b, c \in \mathbb{R}^+$ are parameters and $s \in [0,1]$ is the scaling factor we use for data compression (i.e., $s = 0.95$ means the data size of the compressed image will be roughly 95% that of its original data size).

We will use the same compression method of [53] and the Gompertz approximation to control the compression level for image transmissions through the network during offloading to edge or cloud server. The goal is change the compression level to minimize the latency-accuracy metric of our system.

3.3.3 CODE Problem Definition

Here, we define the ***C**ompression and **O**ffloading **D**ecisions on the **E**dge* (CODE) problem as a *mixed integer nonlinear program* (MINLP). CODE jointly considers offloading and compression decisions. Thus, we consider 4 decision variables: 3 binary offloading decisions (**x**, **y**, **z**) and 1 continuous compression decision (**s**). First, we let decision variable $\mathbf{x} \triangleq (x_i)_{\forall i \in \mathcal{U}} = 1$ iff request i is processed locally using the requesting user's own local hardware resources, 0 otherwise. Second, we let decision variable $\mathbf{y} \triangleq (y_{ij})_{\forall i \in \mathcal{U}, j \in \mathcal{E}} = 1$ iff request i is processed by edge server j, 0 otherwise. Third, we let a set of decision variables $\mathbf{z} \triangleq (z_i)_{\forall i \in \mathcal{U}} = 1$ iff request i is processed on the remote central cloud, 0 otherwise. Finally, we consider a decision variable $\mathbf{s} \triangleq (s_i)_{\forall i \in \mathcal{U}} \in [0,1]$ which corresponds to the compression rate used to compress the data associated with request i prior to processing (i.e., s_i represents the size of request after compression). Below, we present the definition of the CODE problem, dubbed as $\mathcal{P}\mathbf{1}$ for short:

$$\mathcal{P}\mathbf{1}: \underset{\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{s}}{\text{minimize}}\ L - \alpha A \tag{3.2}$$

$$\text{subject to } L \leq L_{\max} \tag{3.2a}$$

$$A \geq A_{\min} \tag{3.2b}$$

$$x_i + \Big(\sum_{j \in \mathcal{E}} y_{ij} \Big) + z_i = 1 \qquad \forall i \in \mathcal{U} \tag{3.2c}$$

$$\sum_{i \in \mathcal{U}} y_{ij} \leq v_j^e \qquad \forall j \in \mathcal{E} \tag{3.2d}$$

$$\sum_{i \in \mathcal{U}} z_i \leq v^c \tag{3.2e}$$

$$x_i, y_{ij}, z_i \in \{0,1\} \qquad \forall i \in \mathcal{U}, j \in \mathcal{E} \tag{3.2f}$$

$$0 \leq s_i \leq 1 \qquad \forall i \in \mathcal{U} \tag{3.2g}$$

The Pareto objective, defined in Eq. (3.2), aims to minimize the trade-off between average latency, L, and average accuracy, A (with α being a tunable hyperparameter). Note, both L and A are functions

of offloading (**x**, **y**, **z**) and compression decisions (**s**). Constraint (3.2a) ensures that the average latency is less than or equal to the maximum allowed latency ($L_{\max}$). Similarly, Constraint (3.2b) guarantees that the average accuracy is greater than or equal to the minimum required accuracy ($A_{\min}$). Constraint (3.2c) guarantees that each request will be processed locally, by 1 edge server, or the central cloud. Constraints (3.2d) and (3.2e) guarantee that the admission capacity of the edge servers and the central cloud server are not exceeded. Note, requests that cannot be processed on an edge server or the central cloud due to the Constraints (3.2d) and (3.2e) will be processed locally. The last two constraints ensure that the decision variables are in the defined range.

CODE Problem Complexity. A robust proof of NP-hardness is beyond the scope of this chapter. With the problem being a *mixed integer nonlinear program* (MINLP), it has been shown that both convex and non-convex MINLPs in general are NP-*hard* [94]. Hence, we assume that the CODE problem is difficult to directly solve for the optimal solution.

3.3.4 Defining Average Accuracy & Latency

Accuracy Definition

Each entity in the system (i.e., user devices, edge servers, and central cloud) hosts an ML model to perform some task (namely, image classification). However, the models can vary across these entities and thus it follows that the accuracy provided by these entities varies as well. Thus, we assume that we have fitted Gompertz functions to estimate the accuracy of the ML model hosted at each entity in the system similar to our initial results in [53]. Using these fitted functions and the compression decisions, **s**, we can compute the average accuracy, A, which is used in the objective for $\mathcal{P}\mathbf{1}$. Its definition is found below:

$$A = \frac{1}{|\mathcal{U}|} \sum_{i \in \mathcal{U}} \left(x_i g_i^u(1) + \sum_{j \in \mathcal{E}} y_{ij} g_j^e(s_i) + z_i g^c(s_i) \right) \tag{3.3}$$

where $g_i^u(\cdot)$ is the Gompertz function fitted for user i's local ML model, $g_j^e(\cdot)$ is the Gompertz function fitted for the ML model hosted at edge server j's, and $g^c(\cdot)$ is the Gompertz function fitted for the ML model hosted at the central cloud. Note that no compression is ever performed if the request is processed locally (i.e., $g_i^u(1)$). The definitions of the individual Gompertz functions are based on fitted parameters and Eq. (3.1). These parameters are provided in §3.5.2.

Latency Definitions

The latency incurred to complete a request i is the sum of both transmission/communication latency and computation latency. In $\mathcal{P}\mathbf{1}$, one of the goals is to minimize the *average latency*, L, by making decisions related to compression and offloading. As such, latency is a function of both compression and offloading decisions. The definition we consider for average latency can be found below:

$$L = \frac{1}{|\mathcal{U}|} \sum_{i \in \mathcal{U}} \left(x_i L^u(i) + \sum_{j \in \mathcal{E}} y_{ij} L^e(i,j) + z_i L^c(i) \right) \tag{3.4}$$

where $L^u(i)$ is the latency to locally process request i, $L^e(i,j)$ is the latency to process request i on edge server j, and $L^c(i)$ is the latency to process request i on the central cloud server.

Local Latency. If a task is processed locally, then latency is only a function of local compute latency since no data transmission is needed to process the task on another device. Thus, we define local latency on the user-side for user i's request, $L^u(i)$, as the following,

$$L^u(i) = \frac{c_i}{f_i^u} \tag{3.5}$$

where c_i is the CPU clock cycles needed to process user i's request and f_i^u is the CPU frequency of user i's device.

User-to-Edge Latency. Tasks processed on the edge require both communication and computation to be completed. First, the user i's request must be transmitted to their covering edge server, w_i, and either processed there or offloaded to another edge server through device-to-device communication channels. Once the request is received by the final edge server, it will be processed, thus incurring additional compute latency. Thus, we define the *user-to-edge latency*, $L^e(i,j)$, as the latency for user i's request to be communicated and processed by edge server j. It is formally presented below,

$$L^e(i,j) = \begin{cases} \frac{s_i d_i}{R_i^u} + \frac{c_i}{f_j^e} & \text{if } j \equiv w_i \\ \left(\frac{s_i d_i}{R_i^u} + \frac{s_i d_i}{R_{w_i j}^e} \right) + \frac{c_i}{f_j^e} & \text{otherwise.} \end{cases} \tag{3.6}$$

The first case occurs when j is user i's covering edge server (w_i). In the first case, only one communication hop and processing is needed — where R_i^u is the data transmission rate from user i to its covering edge server and f_j^e is the CPU frequency on edge server j. The latter case covers the scenario where j is not user i's covering edge server, thus requiring an additional hop of communi-

cation. Here, we note that $R^e_{w_i j}$ is the bit rate between edge servers w_i and j and d_i represents the original size of data in request i.

User-to-Edge-to-Cloud Latency. Finally, we consider the *user-to-edge-to-cloud* latency, $L^c(i)$, for any task i that is offloaded to the central cloud server. It is defined below:

$$L^c(i) = \left(\frac{s_i d_i}{R^u_i} + \frac{s_i d_i}{R^c_{w_i}}\right) + \frac{c_i}{f^c}. \tag{3.7}$$

where $R^c_{w_i}$ is the bit-rate between user i's covering edge server and the central cloud and f^c is the CPU frequency at the central cloud.

3.4 Problem Decomposition & Heuristic Design

Because $\mathcal{P}\mathbf{1}$ (provided in §3.4) is a nonlinear optimization, it is hard to solve. As such, rather than solve it directly, we instead choose to decompose it into two sub-problems, $\mathcal{P}\mathbf{1.1}$ and $\mathcal{P}\mathbf{1.2}$, that separately focus on the offloading decisions and compression decision (respectively). Further, we will solve both sub-problems back-to-back using a *block coordinate descent method* [88] to attain a near-optimal solution for $\mathcal{P}\mathbf{1}$. We define the decomposed sub-problems for $\mathcal{P}\mathbf{1}$ below:

$$\mathcal{P}\mathbf{1.1}: \underset{\mathbf{x},\mathbf{y},\mathbf{z}}{\text{minimize}}\ L - \alpha A \tag{3.8}$$

$$\text{subject to Constraints (3.2a)-(3.2f)} \tag{3.8a}$$

$$\mathcal{P}\mathbf{1.2}: \underset{\mathbf{s}}{\text{minimize}}\ L - \alpha A \tag{3.9}$$

$$\text{subject to Constraints (3.2a), (3.2b), (3.2g)} \tag{3.9a}$$

$$A \geq 0.368 \cdot A^p_{\max} \quad (\forall p \in \{u, e, c\}) \tag{3.9b}$$

where $A^p_{\max}$ represents the maximum provided accuracy by the user layer ($p = u$), edge layer ($p = e$), and cloud layer ($p = c$). Since the Gompertz function has both concave and convex parts, we introduce the constraint (3.9b) to guarantee that we only consider the concave part, which is convex in the negative form. Because the objective of this problem is convex, the constraints should be concave — meaning a nonlinear solver can solve $\mathcal{P}\mathbf{1.2}$ directly.

Algorithm 4: Proposed PCODE Algorithm

Input : Input parameters to CODE, $\lambda \geq 2$ (int)
Output: Offloading/compression decisions ($\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$, $\mathbf{s}$).

Init decision variables $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$, $\mathbf{s}$;
Init total latency $\sigma_L \leftarrow 0$, total accuracy $\sigma_A \leftarrow 0$;
Init total number of processed requests $\eta \leftarrow 0$;
Init $counter^c \leftarrow 0$, $counter^e_j \leftarrow 0$ $(\forall j \in \mathcal{E})$;
$\mathbf{C} \leftarrow$ row-vect. of λ evenly-spaced values from 0 to 1;
foreach *request* $i \in \mathcal{U}$ *in a random order* **do**
    `/* Consider local processing */`
    $\pi^*_u \leftarrow \mathbb{E}[\text{obj}^u_i]$;
    $x_i \leftarrow 1$, $s_i \leftarrow 1$;
    $L_i \leftarrow L^u(i)$, $A_i \leftarrow g^u_i(1)$;
    `/* Consider offloading to edge */`
    foreach $j \in \mathcal{E}$ **do**
        if $counter^e_j < v^e_j$ **then**
            $\psi^*_{ej} \leftarrow \arg\min_{s \in \mathbf{C}} \mathbb{E}[\text{obj}^e_{ij}|s]$;
            $\pi^*_{ej} \leftarrow \mathbb{E}[\text{obj}^e_{ij}|\psi^*_{ej}]$;
        else
            $\pi^*_{ej} \leftarrow \infty$;
    $j^* \leftarrow \arg\min_{j \in \mathcal{E}} \pi^*_{ej}$;
    if *edge offloading is better, i.e.,* $\pi^*_{ej^*} < \pi^*_u$ **then**
        $y_{ij^*} \leftarrow 1$, $s_i \leftarrow \psi^*_{ej}$;
        $x_i \leftarrow 0$;
        $L_i \leftarrow L^e(i, j^*)$, $A_i \leftarrow g^e_{j^*}(\psi^*_{ej^*})$;
    `/* Consider offloading to cloud */`
    if $counter^c < v^c$ **then**
        $\psi^*_c \leftarrow \arg\min_{s \in \mathbf{C}} \mathbb{E}[\text{obj}^c_i|s]$;
        $\pi^*_c \leftarrow \mathbb{E}[\text{obj}^c_i|\psi^*_c]$;
        if *cloud offloading is best, i.e.,* $(\pi^*_c < \pi^*_u) \wedge (\pi^*_c < \pi^*_{ej^*})$ **then**
            $z_i \leftarrow 1$, $s_i \leftarrow \psi^*_c$;
            $x_i \leftarrow 0$, $y_{ij^*} \leftarrow 0$;
            $L_i \leftarrow L^c(i)$, $A_i \leftarrow g^c(\psi^*_c)$;
    Increment *counter* variables for edge and cloud based on offloading decision;
    $\sigma_L \leftarrow \sigma_L + L_i$, $\sigma_A \leftarrow \sigma_A + A_i$, $\eta \leftarrow \eta + 1$;
return $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$, $\mathbf{s}$;

3.4.1 Efficient Algorithm Design

We propose an efficient algorithm to solve $\mathcal{P}\mathbf{1}$. This algorithm approaches $\mathcal{P}\mathbf{1}$ in an iterative manner by considering each user request $i \in \mathcal{U}$ and greedily makes offloading and compression decisions based on the expected gain to the objective of $\mathcal{P}\mathbf{1}$. The pseudocode for the proposed algorithm is presented in Algorithm 4. Before explaining the pseudo code line-by-line, we introduce some mathematical definitions on which PCODE is based upon. First, to adhere to the $L_{\max}$ and $A_{\min}$

constraints (i.e., constraints (3.2a) and (3.2b)), our algorithm uses a *moving average* that tracks the expected average latency and average accuracy for requests as decisions are made. For the moving average, we denote the total latency and total accuracy by σ_L and σ_A, respectively. Next, we denote the number of requests processed thus far by η.

Approximating objective values

The proposed algorithm relies on approximating how an offloading and compression decision for a single user will affect the global objective using a moving average technique. As such, we define how these are approximated. Requests can be processed in 3 ways: local processing, edge-based processing, and cloud-based processing. First, the estimated objective if a request i is processed locally is computed by

$$\mathbb{E}[\text{obj}_i^u] \triangleq \frac{\sigma_L + L^u(i)}{\eta+1} - \alpha \cdot \frac{\sigma_A + g_i^u(1)}{\eta+1} \tag{3.10}$$

where σ_L is the summation of latency across all requests processed thus far based on prior offloading/compression decisions, σ_A is similar but for the summation of accuracies for previously processed requests, and η is the number of previously processed requests. Next, we approximate the global objective if a request i is offloaded to edge server j using compression s by

$$\mathbb{E}[\text{obj}_{ij}^e|s] \triangleq \begin{cases} \frac{\sigma_L + L^e(i,j|s_i=s)}{\eta+1} - \alpha \cdot \frac{\sigma_A + g_j^e(s_i=s)}{\eta+1} & I^e(i,j) \\ \infty & \text{otherwise} \end{cases} \tag{3.11}$$

where $I^e(i,j)$ is an indicator function such that $I^e(i,j) = 1$ if and only if $\left(\frac{\sigma_L + L^e(i,j|s_i=s)}{\eta+1} \leq L_{\max}\right) \wedge \left(\frac{\sigma_A + g_j^e(s_i=s)}{\eta+1} \leq A_{\min}\right)$, 0 otherwise. Finally, we approximate the global objective if a request i is offloaded to the central cloud using compression s by

$$\mathbb{E}[\text{obj}_i^c|s] \triangleq \begin{cases} \frac{\sigma_L + L^c(i|s_i=s)}{\eta+1} - \alpha \cdot \frac{\sigma_A + g^c(s_i=s)}{\eta+1} & I^c(i) \\ \infty & \text{otherwise} \end{cases} \tag{3.12}$$

where $I^c(i)$ is a binary indicator function such that $I^c(i) = 1$ if and only if $\left(\frac{\sigma_L + L^c(i|s_i=s)}{\eta+1} \leq L_{\max}\right) \wedge \left(\frac{\sigma_A + g^c(s_i=s)}{\eta+1} \leq A_{\min}\right)$, $I^c(i) = 0$ otherwise.

Stepping through the Algorithm

PCODE's input includes the system parameters and an integer $\lambda \geq 2$ which is used to generate a $(1 \times \lambda)$-row vector ($\mathbf{C}$) of evely-spaced values from 0 to 1 (e.g., if $\lambda = 3$ then $\mathbf{C} = [0.0, 0.5, 1.0]$). $\mathbf{C}$ is

used as a quantized compression value search space. Larger λ values increase PCODE's complexity but provides more space to find good compression decisions for PCODE. With that said, PCODE starts by initializing decision variables and other supplemental variables (lines 1-5). Line 6 starts a loop that iterates through each request $i \in \mathcal{U}$ in a random order. At the start of the loop, lines 7-9 approximate the gain towards the objective for processing request i locally. Then, lines 10-16 iteratively do the same process but for each of the edge servers $j \in \mathcal{E}$ and identifies the edge server that maximizes the approximated objective value for request i (line 16). On lines 17-20, PCODE determines if processing request i on the edge is better for the objective than local processing. If so, then offloading and compression decisions are changed accordingly (lines 18-20). Then, lines 21-27 do the same for cloud-based processing and change offloading and compression decisions if processing request i is shown to be the best choice (lines 24-27). Lines 28 and 29 then update the variables maintained throughout the loop to approximate the objective and respect admission capacities.

Proposition 4. *PCODE is a polynomial-time algorithm with an asymptotic runtime complexity of* $O(\lambda \cdot |\mathcal{U}| \cdot |\mathcal{E}|)$.

Proof. This can be seen simply be first noting that the outer loop (lines 6-29) takes place exactly $|\mathcal{U}|$ times (i.e., number of requests). For each iteration through this outer loop, PCODE also iterates through each of the edge servers (i.e., a multiplicative of $|\mathcal{E}|$) to consider edge processing. Finally, on line 12 (within the loop iterating through $\mathcal{E}$), $\arg\min(\cdot)$ iterates over the compression space which has λ elements (as per its definition). Since the considering local and cloud processing only occurs once in each iteration through the main outer loop, their complexity is constant and does not contribute to the overall runtime. Thus, the runtime is $O(\lambda \cdot |\mathcal{U}| \cdot |\mathcal{E}|)$. This concludes the proof. □

The complexity of optimal solution due to having a continuous decision variable (**s**) cannot be exactly calculated. We have observed some cases of 9.5 hours running time, while our proposed PCODE algorithm runs in 0.7 seconds, for the same test.

3.5 Experimental Design

3.5.1 Benchmark Algorithms/Heuristics

We demonstrate the efficacy of our proposed algorithm (presented in §30) against the following benchmarks:

Optimal (OPT)

It solves $\mathcal{P}\mathbf{1.1}$ and $\mathcal{P}\mathbf{1.2}$ back-to-back over several iterations using *block coordinate descent* [88] method which minimizes the objective with respect to one block at a time while the other block is fixed.

Optimal_2 (OPT_2)

Its approach is similar to OPT with this difference that it first solves $\mathcal{P}\mathbf{1.2}$ and then $\mathcal{P}\mathbf{1.1}$. For both OPT and OPT_2, we stop running each $\mathcal{P}\mathbf{1.1}$ and $\mathcal{P}\mathbf{1.2}$ back-to-back when either the results in terms of objective converges or a threshold of back-to-back running passes which is set to 10, here, due to computationally expensive cost of running these two algorithms. So, the solver might not be able to find the optimal solution in some cases.

No-Compression (NC)

This heuristic applies no compression but will apply optimal offloading. In short, full image data size is used (i.e., $s_i = 1$ $(\forall i \in \mathcal{U})$) and one iteration of $\mathcal{P}\mathbf{1.1}$ is solved.

Fixed-Compression-50 (FC)

It solves the offloading problem with the assumption that compression is allowed only at a fixed size of $s_i = 0.5$ $(\forall i \in \mathcal{U})$.

Random-Serving-Optimal-Compression (RSOC)

It provides near-optimal compression solution using a similar approach to the PCODE by searching the compression space assuming random offloading decisions. It randomly selects a location (user/edge/cloud) to process requests the requests. If the location is either an edge server or the cloud server and it has enough admission capacity, it selects the minimum possible data size which minimizes the objective; else, it selects another server randomly and repeats the process again.

Random-Serving-Random-Compression (RSRC)

It randomly selects one of the servers (user/edge/cloud). If the server is either edge server or cloud server and it has enough admission capacity, it selects a random size of compression from the range of $(0,1]$; else, it selects another server randomly and repeats the process again.

Worst-case (W)

This algorithm provides a worst case bound for the objective value. It works similar to the PCODE algorithm with this difference that it always selects the maximum objective value, i.e., maximizing the average latency and minimizing the average accuracy.

3.5.2 Environment Setup

We use the Python programming language to simulate the problem. For solving the $\mathcal{P}\mathbf{1.1}$ and $\mathcal{P}\mathbf{1.2}$ problems, we use the Pyomo API with glpk and ipopt solvers [95]. We consider a simulation environment with the following setup. We simulate the accuracy of DL models using a Gompertz function for the cloud server, edge servers, and users' devices. These are used to predict the DL model's accuracy based on compression decisions (see §3.3.2). The Gompertz arguments used for the DL model placed on the cloud server, $a = 0.95$, $b = 20$, and $c = 1$; for the DL model placed on the edge servers, $a = 0.70$, $b = 6$, and $c = 18$; and for the DL model placed on the user devices, $a = 0.45$, 50, and $c = 20$. These are similar to the curves fitted for state-of-the-art DL models in our prior work [53]. Next, the CPU clock cycles to process request i are uniformly sampled from $c_i \in [150,156]$ $(\forall i \in \mathcal{U})$. The size of request i's data (in bytes) is uniformly sampled from $d_i \in [150,156]$ $(\forall i \in \mathcal{U})$. The data transmission rate (bytes/sec) from each user i to their covering edge server, w_i, is uniformly sampled from $R_i^u \in [150,175]$ $(\forall i \in \mathcal{U})$. Then, the CPU frequency of user i's device is uniformly sampled from $f_i^u \in [5,8]$ $(\forall i \in \mathcal{U})$. Data transmission rate (bytes/sec) for D2D communication between edge servers j and j' is uniformly sampled from $R_{jj'}^e \in [200,206]$ $(\forall j, j' \neq j \in \mathcal{E})$ and each edge server has a CPU frequency uniformly sampled from $f_j^e \in [500,506]$ $(\forall j \in \mathcal{E})$. Further, admission capacities for edge servers (i.e., number of requests that can be served in an instance of the problem) are randomly sampled from $v_j^e \in [5,7]$ $(\forall j \in \mathcal{E})$. Finally, data transmission rates (btyes/sec) from the edge servers to the cloud server are uniformly sampled from $R_j^c \in [4,6]$ $(\forall j \in \mathcal{E})$, the CPU frequency of the cloud server is uniformly sampled from $f^c \in [10000,10007]$ in each trial. The cloud's admission capacity is fixed at $v^c = 60$.

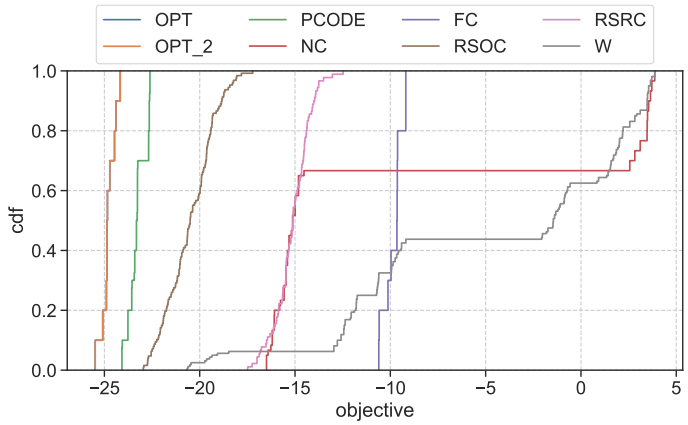

Figure 3.2: CDF of Objectives (OPT and OPT_2 are overlapping and are the leftmost curves)

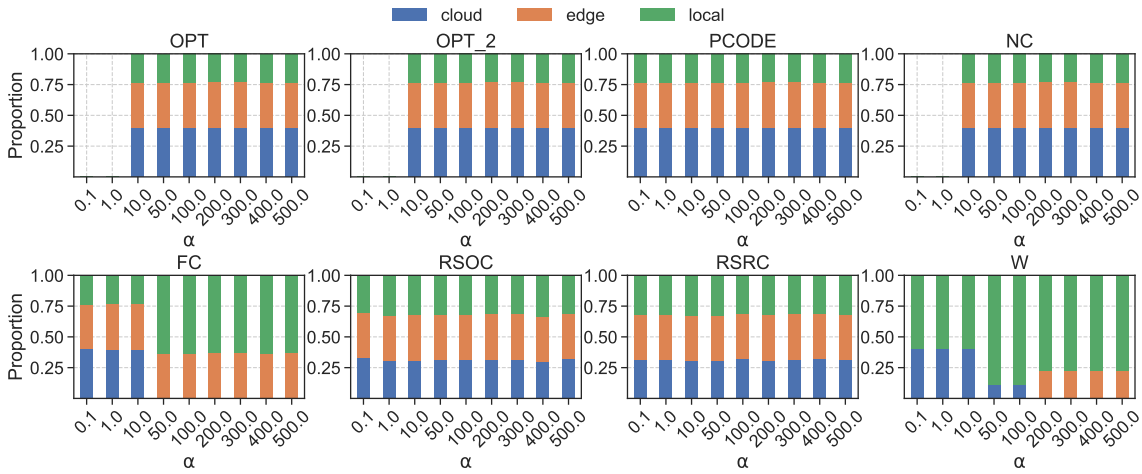

Figure 3.3: Impact of α on the proportion of served requests by each layer

We fix the seed and repeat each test 10 times to remove the effect of randomness. The total number of user requests, $|\mathcal{N}|$, is set to 150 where each of requests is associated with one user and they are randomly connected to the edge servers. The number of edge servers, $|\mathcal{M}|$, is set to 10. We fix the α to 50, $L_{max} = 50$, and $A_{min} = 0.5$, unless otherwise stated. We set the λ, the size of vector C of generated compressed data, for both our proposed algorithm and RSOC equal to 11.

3.6 Results

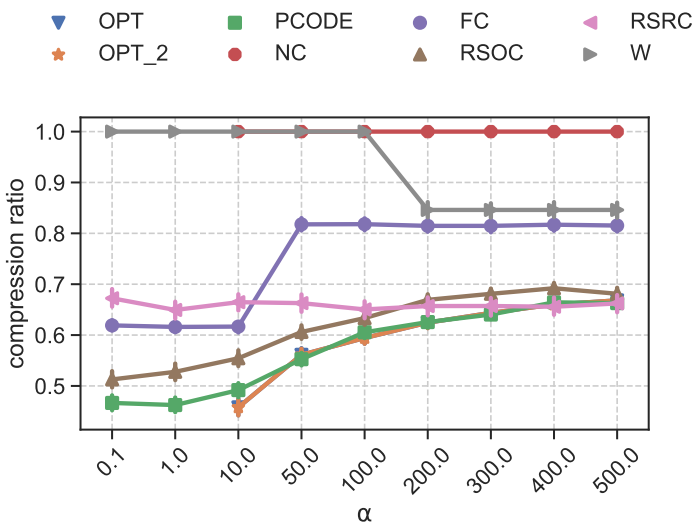

Figure 3.4: Impact of α on the compression decision

3.6.1 Impact of $A_{\min}$ & $L_{\max}$ on the Objective Value

We change the $A_{\min}$ from 40% to 70% and & $L_{\max}$ from 20 to 50 ms to evaluate the effect of QoS metrics–average latency and average accuracy– on the objective value. We set the α equal to 50. Fig. 3.2 represents the Cumulative Distribution Function (CDF) of the objective value of the proposed problem obtained by each algorithm when A_{min} and L_{max} are changed. The OPT and OPT_2 converge to the same results. Therefore, they show same values on the plot. As shown in Fig. 3.2, the proposed algorithm provides near optimal results in terms of objective value even when A_{min} and L_{max} are changed while the performance of baseline algorithms decreases when A_{min} and L_{max} are changed.

3.6.2 Impact of α on Compression Decisions (s) and the Objective Value

We fix the $L_{\max}$ to 50 and $A_{\min}$ to 0.5 and change the α in the range of 0.1 to 500 to evaluate the effect of α on compression decisions and the objective value. The objective value of all algorithms based on different α values are provided in Table. 3.1. The results imply that PCODE achieves near optimal solutions. Given the results presented in Table. 3.1, our proposed PCODE algorithm can empirically achieve in average 0.958-approximation of the optimal solution provided by OPT and OPT_2.

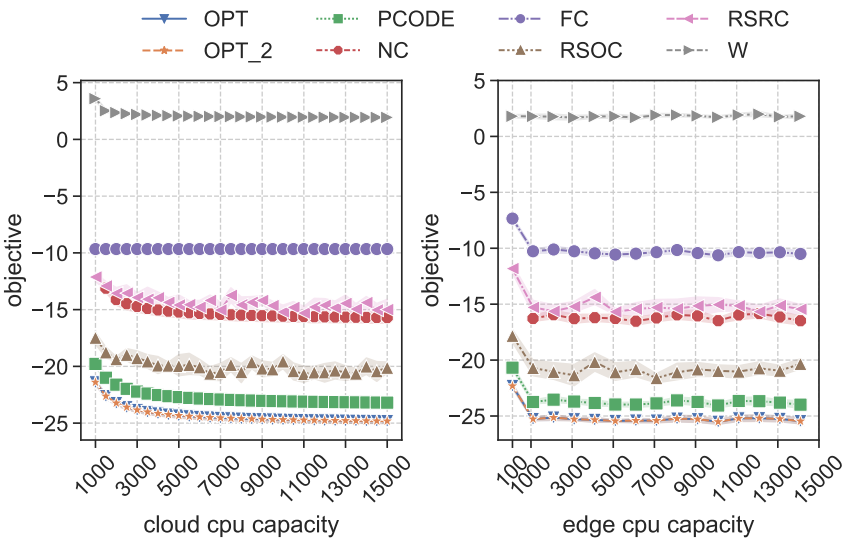

Figure 3.5: Impact of edge servers and cloud server CPU on objective value

Table 3.1: The average objective value of all algorithms based on different α values (note that "-" means that algorithm did not find valid solution for that case.)

	α Values				
Algorithm	1.0	10	50	100	200
OPT	-	3.231	-24.779	-61.231	-135.037
OPT_2	-	3.231	-24.779	-61.231	-135.037
PCODE	10.631	4.921	-23.260	-59.705	-133.419
RSOC	11.959	6.411	-20.301	-54.546	-124.137
RSRC	14.263	9.170	-14.752	-44.173	-102.950
NC	-	14.056	-15.547	-52.739	-127.165
FC	13.408	6.483	-9.820	-36.756	-91.287
W	29.063	24.167	1.750	-23.202	-79.416

The effect of α on the compression decision of requests is shown in Fig. 3.4 where the y-axis represents the average compression ratio of all requests and the x-axis represents different α values. When α increases, the accuracy dominates the latency and the compression ratio, s_i, is increased which is intuitive given that larger data size provides better accuracy. Fig. 3.3 represents the portion of requests served by each layer–cloud, edge, and users. As shown, the optimal solutions and the proposed algorithm provide a trade-off between the amount of resources used by each layer in the network while most of baseline algorithms do not. Although the RSOC and RSRC provide a trade-off between a portion of consumed resources in the networks, their objective value is worse than our proposed algorithm, PCODE, based on Table. 3.1. Additionally, the solver cannot find a solution for OPT and OPT_2 when $\alpha < 10$ because we limited the number of back-to-back running of $\mathcal{P}\mathbf{1.1}$ and $\mathcal{P}\mathbf{1.2}$. While our proposed algorithm can find a solution for these cases in less than a second.

3.6.3 Impact of Hardware Resources

We changed the CPU capacity of both the cloud layer and the edge layer. We fixed the edge CPU capacity across all edge servers. Fig. 3.5 represents the effect of cloud server CPU and edge servers CPU on the objective value of all algorithms. When either the cloud server CPU or the edge servers CPU increases, the objective improves. Fig. 3.5 implies that the objective value of our proposed PCODE algorithm is close to OPT and OPT_2 in all cases of different CPU capacities for both edge servers and the cloud server.

3.7 Conclusions

This chapter studies joint offloading and compression decisions in a 3-tier user-edge-cloud system considering limited communication and computation capacity of the devices. We proposed a Mixed Integer Nonlinear Program (MINLP) to solve the proposed problem. We decompose the proposed MINLP into two sub-problems: one that solves the offloading sub-problem and another that solves the the compression sub-problem with one set of continues decision variables. The MINLP is NP-hard. Therefore, we propose a heuristic algorithm, namely PCODE, to solve the problem. We compare our proposed PCODE algorithm against several baselines in terms of objective value and the final decisions of each algorithm regarding both the compression and offloading problems using extensive simulation. Our results show that PCODE can match the solution found by the optimization solver by an 0.958-approximation on average. In future works, we will focus on a real-world implementation of this work.

4

A Framework for Edge Intelligent Smart Distribution Grids via Federated Learning

©2021 IEEE. Reprinted with permission from Nathaniel Hudson, "A Framework for Edge Intelligent Smart Distribution Grids via Federated Learning," 2021 International Conference on Computer Communications and Networks (ICCCN), July 2021,
DOI: 10.1109/ICCCN52240.2021.9522360

Recent advances in distributed data processing and machine learning provide new opportunities to enable critical, time-sensitive functionalities of smart distribution grids in a secure and reliable fashion. Combining the recent advents of edge computing (EC) and edge intelligence (EI) with existing advanced metering infrastructure (AMI) has the potential to reduce overall communication cost, preserve user privacy, and provide improved situational awareness. In this chapter, we provide an overview for how EC and EI can supplement applications relevant to AMI systems. Additionally, using such systems in tandem can enable distributed deep learning frameworks (e.g., federated learning) to empower distributed data processing and intelligent decision making for AMI. Finally, to demonstrate the efficacy of this considered architecture, we approach the non-intrusive load monitoring (NILM) problem using federated learning to train a deep recurrent neural network architecture in a 2-tier and 3-tier manner. In this approach, smart homes locally train a neural network using their metering data and only share the learned model parameters with AMI components for aggregation. Our results show this can reduce communication cost associated with distributed learning, as well as provide an immediate layer of privacy, due to no raw data being communicated to AMI components. Further, we show that FL is able to closely match the model loss provided by standard centralized

deep learning where raw data is communicated for centralized training.

4.1 Introduction

As the Internet of Things (IoT) matures, more environments originally designed to be disjoint are becoming increasingly interconnected. One such environment generating much excitement is that of smart grids. The vision and realization of various aspects of smart grids have been underway for a couple of decades now. An important sub-component of smart grids is advanced metering infrastructure (AMI). AMI is an integrated system of smart meters, communication networks, and data management systems to provide two-way communication between utility providers and customers. The advent of AMI provides an opportunity to allow for services not previously possible with more conventional power distribution infrastructures. The vast amount of data collected by these systems enables a wide range of applications and can likely be combined by other modes of data to make even smarter decisions related to the distribution grid functions. Such smart distribution grid functions include automated metering of energy consumption, demand response (DR) management [96], dynamic pricing (DP) [97], tampering detection, outage identification, and wide area monitoring systems (WAMS) [98].

Handling and processing the large volume of sensitive data for smart grids, and particularly for AMI applications, require communication- and computational-resource-aware and privacy- and security-conscious solutions. Emerging distributed computing technologies, and more specifically edge computing (EC), can provide computationally-intensive and data-oriented applications closer to where the data are being generated [32]. Moreover, pushing artificial intelligence (AI) to the edge, known as edge intelligence (EI), introduces immense potential for EC to provide computationally-intense models (e.g., deep learning) for smart grids. EC, along with EI, provides a viable ecosystem with countless possibilities to support various smart grid functions and services with ameliorated delay, robust reliability, and improved data privacy/security [22].

In light of stringent data privacy requirements and growing privacy concerns in smart grids, the recent *federated learning* (FL) framework has been considered as a promising EI solution for smart grids [99]. In FL, a machine learning (ML) model (e.g., deep neural network) is trained in a distributed fashion by sending the model updates rather than raw data to the central server for aggregation. This consideration immediately introduces a layer of privacy and has shown to be an effective approach to deep learning for edge systems in terms of communication cost and

performance efficacy [100].

In this chapter, non-intrusive load monitoring (NILM) function in AMI is considered to demonstrate the promising potential of EI and FL in supporting smart grid applications. NILM allows detailed energy sensing from aggregated data (e.g. from smart meters), which is essential for energy management solutions and addressing energy conservation challenge due to increasing energy demands. However, NILM can reveal detailed appliance-specific energy consumption statistics, which can expose privacy and security risks to consumers and, as such, needs to be handled delicately. In this chapter, 2-tier and 3-tier FL frameworks are developed for the NILM function. The performance of the proposed models are evaluated with respect to communication cost and the model training loss. We compare these approaches to a centralized deep learning solution. Our results show that our 2-tier and 3-tier FL approaches show comparable performance to the centralized deep learning approach. However, our 2-tier and 3-tier FL approaches reduce the associated communication cost, while providing some immediate layer of privacy, because raw data is not communicated to central servers in AMI systems.

The presented study along with recent developments in this domain suggest that EI and FL have great potential in addressing data processing challenges in smart grids. However, research and practice on this emerging multidisciplinary domain—EI-enable smart grids—are still in a very early stage. These studies can help bridge the gap between highly dynamic and fast-growing EI and FL domains and smart grids. The marriage of these technologies can ultimately work towards a more reliable, secure, and efficient smart grids. The remainder of the chapter is organized as follows. In §4.2, we present a concise, yet robust overview of AMI systems alongside their benefits and challenges. Then, in §4.3 we discuss edge computing and edge intelligence and how these concepts can supplement AMI specifically. §4.4 describes our 2-tier and 3-tier FL approach considered for the NILM problem. We then discuss experimental design in §4.5. We discuss the findings of our experimental results in §5.6. Finally, in §5.8, we present our conclusions and discuss open challenges, problems, and points of interest related to using EC and EI for AMI systems.

4.2 Smart Distribution Grids & Advanced Metering Infrastructure

4.2.1 AMI Overview and Requirements

The evolution of smart grid technology as a cyber-physical system with extensive integration of metering devices, such as smart meters, phasor measurement units (PMUs) [101] and micro-PMUs [102, 103], has enabled the utilities to evolve from a traditional bulk centralized power grid to a system capable of distributed generation (DG), monitoring, and control. Such technologies along with their supporting communication and computation platforms provide real-time data and analytics power to provide a more reliable, secure, and efficient operation of the system through situational awareness (SA) [104] functions (e.g., remote asset monitoring, distribution-level oscillation analysis), as well as other applications related to power and services (e.g., customer profiling for demand response, peak leveling and dynamic pricing and more). AMI in particular is a key component of the smart distribution grid that is responsible for collecting and analyzing data to support many of the smart distribution grid functions.

AMI is a collection of technologies consisting of systems, meters, and networks (for two-way communication between utilities and consumers) to monitor, collect and analyze time-stamped consumer and system information such as voltage, power, phase, and load as sequential meter readings [98]. The AMI can expand all the way to the end-user devices connected to the smart meters through *home area networks* (HANs). The data collected from the AMI is stored and analyzed in *meter data management system* (MDMS) which includes long-term storage as well as analytical tools [105].

AMI Communication System Requirements: The communication system is an important component of AMI, which enables two-way communication between the consumer and the utility to transfer the collected data as well as operational commands. Considering the size of the system and the large number of end-users, the communication network is required to support the transfer of a high volume of data. The overhead of communicating this large amount of data (to a centralized server) can introduce latency in time-critical applications and interrupt seamless operation and control. As such, it is important that the communication system supporting AMI provides a reliable transfer that also meets the latency requirements of the AMI functions and applications. The minimum data rate in AMI is considered to be 56 kbps and the maximum acceptable latency to be 2 seconds [106]. However, considering the fast developing technologies, functions and services in smart grids, the communication network should be designed to support future needs with more stringent latency requirements. In general, the AMI communication network can be deployed based

on wired or wireless technologies, such as Power Line Communications, Broadband over Power Line (BPL), landline, or cellular networks [107, 108] considering cost, availability, and support for future expansion and applications.

AMI Privacy and Security Requirements: The consumer data collected by the smart meters enables various applications including consumer profiling, adaptive pricing, and energy theft detection; however, it also bears personal information about consumers, which needs to be protected [109]. As such, the AMI subsystems including its communication network should enable controlling data access, preserving the confidentiality of data, and ensuring data authenticity and precision. Various measures including hardware security technologies for preventing tampering with smart meters, data encryption technologies, and communication network security solutions against Denial-of-Service (DoS), False Data Injection (FDI) attacks [110, 111] have been considered in AMI to support these requirements. A review of efforts in addressing the security and privacy challenges in smart grids and AMI has been presented in [111, 112].

4.2.2 AMI Functions and NILM

The data generated and processed in AMI and smart grids can fuel a vast set of smart decision making processes and functions in smart grids, which are essential for their reliable, secure and efficient operation. Examples of such data-driven functions include: wide area monitoring and control and situational awareness [113] for voltage/power profiling [114], oscillation detection, topology identification [115], and cyber and physical event detection [116, 117, 118], and anomalous activity detection [119, 120, 121] as well as, customer profiling and load monitoring (e.g., NILM) for demand response, peak leveling and dynamic pricing [122]. As NILM is the application of focus in this study, we present a brief overview of this function and the existing work in this domain next.

NILM (also known as load disaggregation) is the process used for appliance load monitoring that can support many other energy management solutions in smart grids such as energy conservation, abnormality detection, dynamic load scheduling, load shedding, and dynamic energy pricing. It provides fine-grained energy monitoring without the need for deploying smart power outlets on every device by analyzing the aggregated data collected from smart meters. NILM can reveal appliance-specific energy consumption statistics and consumer behavior and as such privacy is a major concern in this function. To process the large volume of data from smart meters for the purpose of NILM, while preserving privacy requirements various techniques have been proposed in

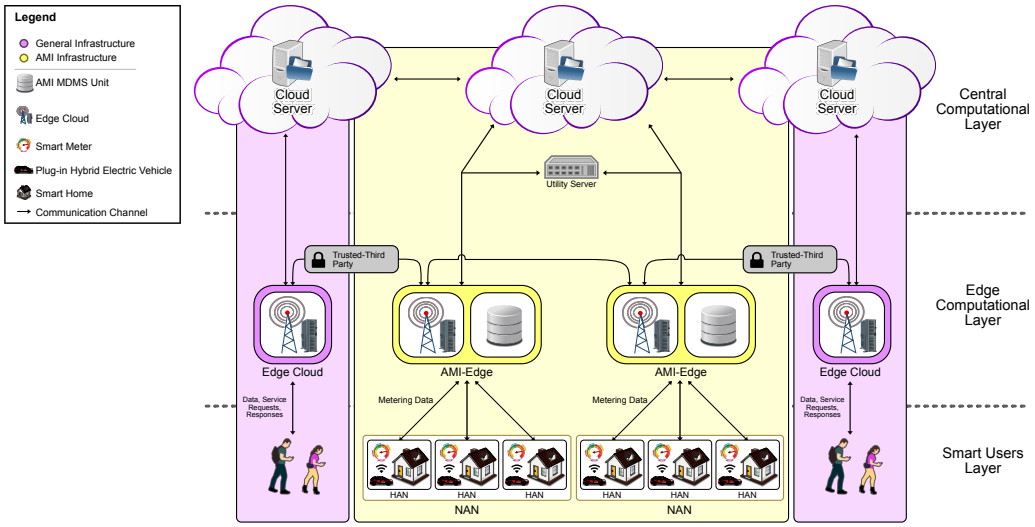

Figure 4.1: An overview of EI-AMI architecture. AMI infrastructure (in yellow) are resources deployed specifically for the EI-AMI architecture and is a subcomponent of the distribution grid. General infrastructure (in purple) are EC resources deployed for general use but can be leveraged through communication channels to supplement the AMI infrastructure.

the literature [123, 124]. Among such techniques include approaches based on Hidden Markov Model (HMM) [125], Graph Signal Processing (GSP) [126], Deep Learning [127, 128, 129, 130], and classification based approaches based on load signature learned from historical data [123, 124]. Among such techniques, deep learning-based approaches are gaining more popularity due to the availability of a large volume of data and new supporting platforms and frameworks for their development in distributed and privacy-preserving manners. Specifically, the emerging distributed computing technologies, EC and EI including FL [131], as discussed in subsequent sections, can provide such supporting platforms not only to support NILM function (as discussed later in §4.5) but also various data-driven AMI applications and their aforementioned requirements.

4.3 Edge Intelligence for Smart Grids

4.3.1 EC Overview and Architecture

Cloud computing can provide a wide array of computationally-intense services (e.g., data analytics) to end users with minimal local hardware resources. With increasing demands on the cloud and emerging delay-sensitive applications (such as AMI functions related to situational awareness), there is a clear need to look beyond the cloud [80]. One promising paradigm is *edge computing* (EC) [53, 32] — also known as fog [18] or micro-cloud computing [20]. EC often considers the deployment

of servers close to the users, known as *edge clouds*, with their own computational resources to serve users while providing reduced latency [132, 133, 33, 134]. EC also does not require all user data to be sent and/or stored in some remote, central cloud server — providing some degree of privacy.

Many EC architectures consider a three-tier architecture of central cloud, edge clouds, and end users [15, 41, 133] where edge clouds can cooperate with the central cloud or other edge clouds to serve user requests [41, 54]. The EC architecture that can support AMI is depicted in Fig.4.1 and a brief description of the EC layers is presented next.

Central Computational Layer

This layer corresponds to the central cloud in EC and the utility central processing server. This layer is tasked with delegating decisions related to other portions of the smart grid system (e.g., billing and distribution). It can also provide additional computational power to edge resources if necessary to serve a service request from a connected component (e.g., a connected smart home). It is not necessary for these central cloud computational resources to be physically adjacent to the utility component for AMI. These resources can be rented through a trusted vendor that provides cloud computing resources.

Edge Computational Layer

This layer is made up of two key components: MDMS units (or data collector units) from AMI and edge clouds from EC. The former component is tasked with the traditional responsibilities of aggregating metering data over a given neighborhood in AMI to perform immediate decisions related to the distribution grid. The latter is tasked with handling service requests and/or collecting data submitted by connected devices. The MDMS units and edge clouds can work together collaboratively. In this regard, two cases can be considered: *(i)* the case where MDMS units and edge clouds are jointly paired with one another and *(ii)* the case where edge clouds are not directly tied with an MDMS unit but communicate with AMI infrastructure through a trusted third party to provide additional resources. In addition, the data collected/generated by and analytics provided by edge clouds can additionally supplement decision making in AMI.

Smart Users Layer

This layer corresponds with a smart home's HAN, equipped with a smart meter and other devices. At each time-step, a smart meter associated with a smart home will communicate metering data to its covering MDMS unit in the Edge Computational Layer. This layer corresponds with IoT devices that interact with edge clouds. These devices are not an explicit part of the architecture, but their existence is worth noting.

4.3.2 Edge Intelligence

The notion of *edge intelligence* (EI) is the deployment of computationally-intensive intelligent models (predominantly machine learning, deep learning, and data analytics) on edge clouds rather than the central cloud. EI provides the same class of benefits to end users provided by EC, such as ameliorated delay and robust scalability. For critical infrastructures, such as AMI, these are necessary benefits. AMI-related data are often sensitive and have associated privacy concerns. It is worth noting that these details are relevant for metering data provided by AMI. However, training EI models using data hosted by end users introduces some challenges with regard to privacy. Additionally, the distribution of data for training among end users are typically *non-independent and identically distributed* (non-iid) — which can pose challenges for training adequate models due to the unbalanced nature of the data. The recent frameworks of FL and *split learning* [135] are promising solutions for distributed learning that addresses these concerns. A recent and comprehensive survey of EI can be found in [22].

4.3.3 Federated Learning

FL was first proposed as a framework for training machine learning models in a distributed, collaborative fashion in client-server environments (e.g., EC) [100]. Under the FL framework, a central server initializes a machine learning model with random model parameters and shares the model with the clients in the system. Over a series of global training rounds, the server selects some portion of clients to perform local training using their local data to update their local model's parameters. These clients then communicate their updated parameters to the server. The server then uses a *federated aggregation* (FA) algorithm to aggregate the collected updated model parameters to then update its global model for redistribution across the clients. This framework provides immediate user/client privacy by not relying on the communication of client data — only the updated model

Algorithm 5: 2-Tier Federated Averaging (FedAvg)

Input : K clients indexed by k, B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Initialize w_0;
foreach *round* $t = 1, 2, \cdots$ **do**
    $m \leftarrow \max(1, C \cdot |\mathcal{K}|)$;
    $S_t \leftarrow$ random subset of m clients from $\mathcal{K}$;
    foreach *client* $k \in S_t$ **do**
        $w_{t+1}^k \leftarrow LocalUpdate(k, w_t)$;
    $w_{t+1} \leftarrow \sum_{k \in \mathcal{K}} \frac{N_k}{N} w_{t+1}^k$
Procedure $LocalUpdate(k, w)$
    $\mathcal{B} \leftarrow$ split $\mathcal{P}_k$ into batches of size B;
    foreach *local epoch* $i = 1, 2, \cdots, E$ **do**
        for *batch* $b \in \mathcal{B}$ **do**
            $w_{t+1} \leftarrow w_t - \eta \nabla l(w_t; b)$;

parameters. Additionally, FA algorithms, such as `FedAvg` [136] and `FedProx` [137], perform well in the face of non-iid data distributions of client data. These benefits address potential concerns related to metering data in AMI (e.g., distribution of metering data across different homes), suggesting that FL can be a promising approach to AMI-related problems (e.g., load forecasting and adaptive pricing). This could allow MDMS units to learn from each other's data while maintaining privacy. More recently, decentralized FL where edge clouds are able to perform FAs among themselves via device-to-device communication channels has also been studied [138, 139]. Additionally, the framework of FL has also been adapted and studied for problems related to the reduction of energy consumption on resource-constrained devices (e.g., IoT devices) in [140].

The added privacy of FL has made it an attractive framework for approaching problems related to smart grids and AMI more specifically. Recently FL has been considered for AMI applications, such as load forecasting [99] and disaggregation/non-intrusive load monitoring [131]. These prior works consider a simple 2-tier ecosystem. Because FL is a flexible framework [138, 139], we can expand upon this assumption by considering the aggregation point to be in different components of the considered architecture. As such, we investigate both 2-tier and 3-tier FL where the latter has an additional aggregation step involving the central utility server.

4.4 FL-Based NILM Model

In this section, we discuss our approach of using FL for the NILM problem (discussed earlier in §4.2.2). First, we briefly describe the deep neural network architecture that we employ for this work and then describe, in detail, how we approach both 2-tier and 3-tier FL. In both approaches, we consider the *federated averaging* (FedAvg) algorithm proposed by McMahan et al. in [136] to perform the task of model parameter aggregation. The pseudocode for this algorithm is provided in Algorithm 5.

4.4.1 Deep Neural Network Model Design

In [127], Kelly et al. chiefly use *long short-term memory* (LSTM) architectures for the NILM problem. LSTMs are a recurrent neural network architecture that has specialized gates to avoid the vanishing and exploding gradient problems. They are especially useful for time-series data. However, the complex nature of LSTMs induce additional training cost in terms of complexity. As such, the more recent *gated recurrent unit* (GRU) has demonstrated comparable performance with reduced complexity [141]. We thus incorporate a simple GRU architecture. Further, we consider the input to our model to be time-series data of length ℓ. Our GRU model $f(\cdot)$ will consider two inputs: an aggregated energy signal $\mathbf{e} = \langle e_i, e_{i+1}, \cdots, e_{i+\ell} \rangle$ and a time signal (in minutes) $\mathbf{t} = \langle t_i, t_{i+1}, \cdots, t_{i+\ell} \rangle$, both of length ℓ. Given these inputs, our model will predict the disaggregated energy signal for a considered appliance in the $(\ell + 1)$ time-step. We leverage dropout to prevent overfitting. Our considered GRU architecture can be found below. For clarity, the model accepts 2 inputs of length ℓ. The energy signal, **e**, is sent to the energy GRU sub-network and the time signal, **t**, is sent to the time sub-network:

- (Energy GRU Sub-Network):
  - GRU(units=64, #layers=2, activation=Tanh)
  - Dense(units=32)
- (Time Sub-Network):
  - Dense(units=32, activation=Tanh)
- Flatten([Outputs of energy and time sub-networks])

Algorithm 6: Proposed 3-Tier Federated Averaging

Input : K clients indexed by k, B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

1 Initialize global model parameters w_0;
2 **foreach** *round* $t = 1, 2, \cdots$ **do**
3 **foreach** *AMI-Edge* $a \in \mathcal{A}$ **do**
4 $\hat{w}_t^a \leftarrow w_t$;
5 $m \leftarrow \max(1, C \cdot |\mathcal{K}^a|)$;
6 $S_t \leftarrow$ random subset of m clients from $\mathcal{K}^a$;
7 **foreach** *client* $k \in S_t$ **do**
8 $w_{t+1}^k \leftarrow LocalUpdate(k, \hat{w}_t^a)$;
9 $\hat{w}_{t+1}^a \leftarrow \sum_{k \in \mathcal{K}^a} \frac{N_k^a}{N^a} w_{t+1}^k$;
10 $w_{t+1} \leftarrow \sum_{a \in \mathcal{A}} \frac{|\mathcal{K}^a|}{|\mathcal{K}|} \hat{w}_{t+1}^a$;

- Dense(64, activation=Tanh)
- Dense(32, output_size=1, activation=Linear)

4.4.2 2-Tier & 3-Tier Federated Learning

For this work, we are interested in considering 3-tier FL in addition to 2-tier FL. The latter is as described in §4.3.3 and pseudocode for this setup using the `FedAvg` [136] algorithm is presented in Algorithm 5. In 2-tier FL, we consider a simple scenario where smart homes $k \in \mathcal{K}$ communicate their updated model parameters w_{t+1}^k to their covering AMI-Edge (see Fig. 4.1). Following the `FedAvg` algorithm, the AMI-Edge will perform a weighted average to aggregate model parameters across all smart homes it covers; the weight is based on the number of data items a smart home has relative to all the smart homes covered by its AMI-Edge. However, a 2-tier approach may not be sufficient. It is intuitive to imagine a relationship between the geography of homes and their respective energy consumption. For instance, homes in more affluent regions of a smart city will likely have more high-energy appliances than homes from less affluent regions. Therefore, the model parameters aggregated by an AMI-Edge covering a specific region may require an additional layer of aggregation to have a more general model. For this, we consider a 3-tier FL approach for NILM. While FL has recently been considered for the NILM problem [131], this marks the first attempt (to the best of our knowledge) to consider an extra layer of aggregation.

The pseudocode for the proposed 3-tier adaptation of the FedAvg algorithm is presented in Algorithm 6. Line 1 initializes the global model, which will be hosted by the central utility server in

the context of AMI. Lines 2-9 then commence the iterative rounds of FL. First, the algorithm iterates through each AMI-Edge $a \in \mathcal{A}$ (lines 3-10) and a downloads the most recent model from the utility server (line 4). The AMI-Edge then selects a random subset of homes it covers, $\mathcal{K}_a$, based on some proportion C (lines 5-6). Each of the randomly selected homes will then train the most recently aggregated model using their own data and update their local model parameters as a result (line 8). Then, in line 9, the AMI-Edge averages the model parameters across all of its covered homes in a weighted fashion via N_k^a/N^a where N_k^a is the number of data items at home k belonging to Edge-AMI a and $N^a \triangleq \sum_{k\in\mathcal{K}} N_k^a$. Finally in line 10, after each Edge-AMI has performed neighborhood-wide aggregation in this round, the central utility server will aggregate the newly updated AMI-Edge models in a weighted fashion via $|\mathcal{K}_a|/|\mathcal{K}|$ where $\mathcal{K}_a$ is the set of smart homes covered by AMI-Edge a and $\mathcal{K}$ is the set of all smart homes (i.e., $\mathcal{K} \triangleq \bigcup_{a\in\mathcal{A}} \mathcal{K}_a$).

4.5 Experiment Design

Here, we describe the design of our evaluation of considering deep learning for the task of NILM on the considered architecture (discussed in §4.3.1). The central aim of this work is to demonstrate the trade-offs between centralized and federated learning of a deep neural network for the NILM problem. These trade-offs are with respect to the trained models' performance and the incurred communication cost associated with each training approach. Further, we consider both a 2-tier or 3-tier approach to FL. It should be noted that FL also has the benefit of providing an immediate layer of privacy. Measuring the degree of privacy is beyond the scope of this work. For a comprehensive survey on the privacy/security benefits of FL, refer to [142].

4.5.1 Data Description

From the Pecan Street dataport [143], we have real-world energy consumption data from 25 actual homes in the New York region over a span of 6 months. The data is collected in 1 sample/minute frequency and contains disaggregated energy consumption signals for a large number of different appliances including distributed generation. For simplicity, we focus specifically on 3 appliances: clothes washer, electric vehicle, and refrigerator. We further pre-processed the data by considering an auxiliary energy signal by summing the energy consumed by each of the 3 appliances at each recorded time-step. This auxiliary signal represents the measurements from smart meters which we used as input to the disaggregation algorithms. Finally, we split 22 of the homes into neighborhood

blocks consisting of 10, 8, and 4 homes. We consider each neighborhood block to be covered by a single Edge-AMI node. The remaining 3 are reserved for testing/validation.

4.5.2 Communication Model

For this evaluation, we are not only interested in the accuracy that an EI model can provide to both home owners and the utility companies but also interested in the network load (or communication cost) associated with the discussed learning approaches. The approaches we have discussed include centralized learning (i.e., all data sent to a central operator before training), 2-tier FL (i.e., learning happens at the smart home level and model aggregation occurs locally at each AMI-Edge node), and 3-tier FL (i.e., each AMI-Edge node's aggregated models are aggregated by the central operator). We now provide a simple, yet intuitive, communication model to analyze the cost of network load affiliated with each of these approaches to gauge the communication benefit of these approaches. As a reminder, $\mathcal{A}$ refers to the set of AMI-Edge nodes, $\mathcal{K}$ refers to the set of all smart homes, and $\mathcal{K}^a$ refers to the set of smart homes directly covered by AMI-Edge $a \in \mathcal{A}$.

Centralized Network Load

First, we consider the standard approach of centralized learning. As mentioned, we assume all data are sent to a central operator in central computational layer of the considered architecture (see Fig. 4.1). As such, all data from a smart home are first communicated to its covering AMI-Edge node AMI-Edge node and then communicated to the central server. We define it below,

$$L_c \triangleq \sum_{a \in \mathcal{A}} \sum_{k \in \mathcal{K}^a} |\mathcal{P}_k| \cdot (\text{hops}_{ka} + \text{hops}'_a) \tag{4.1}$$

where $|\mathcal{P}_k|$ is the size of the data hosted by smart home k, hops_{ka} is the number of hops from smart home k to its covering AMI-Edge a, hops'_a is the number of hops from AMI-Edge a to the central utility server.

2-Tier FL Network Load

In FL, only the parameters for the model being trained are communicated between AMI components (e.g., smart home, AMI-Edge, central utility server). As such, we instead consider the size of the model parameters rather than the size of the raw data hosted at each smart home. For 2-tier FL specifically, we consider each AMI-Edge to aggregate model parameters trained by its covered smart

homes independently of any other system component. Therefore, the model parameters are only ever communicated from the smart homes to their covering AMI-Edge. As such, we define the network load for 2-tier FL, L_f, in Eq. (4.2) below,

$$L_f \triangleq \sum_{a \in \mathcal{A}} \sum_{k \in \mathcal{K}^a} |\text{model}| \cdot \text{hops}_{ka} \tag{4.2}$$

where |model| is the size of the parameters for the model being trained distributedly and hops_{ka} is the number of hops from smart home k to AMI-Edge node a.

3-Tier FL Network Load

In 3-tier FL, we consider an extra step of aggregation. Each of the AMI-Edge nodes will communicate their newly aggregated model parameters to the central utility server to further aggregate them (according to Algorithm 6). As such, we define the network load for 3-tier FL, L'_f, in Eq. (4.3) below,

$$L'_f \triangleq \sum_{a \in \mathcal{A}} \sum_{k \in \mathcal{K}^a} |\text{model}| \cdot (\text{hops}_{ka} + \text{hops}'_a) \tag{4.3}$$

where |model| is the size of the parameters for the model being trained distributedly, hops_{ka} is the number of hops from smart home k to AMI-Edge node a, and hops'_s is the number of hops from AMI-Edge a to the central utility server.

4.6 Results & Discussion

Here, we present and discuss the results of our approach to the NILM problem. The focus of these results is the comparison between FL-based solutions to the NILM problem against centralized deep learning when considering model performance and incurred communication cost/network load.

4.6.1 Network Load Analysis

Here, we evaluate the network load incurred from training machine learning models through centralized, 2-tier federated, and 3-tier federated approaches. To perform this evaluation, we fix the number of hops from each AMI-Edge $a \in \mathcal{A}$ to each of its smart homes $k \in \mathcal{K}^a$ (hops_{ka}). We vary this value over the range $[1, 10]$. We do the same for the number of hops between the central utility server and the AMI-Edge nodes, hops'_a. For this chapter, we use the real-world measurements

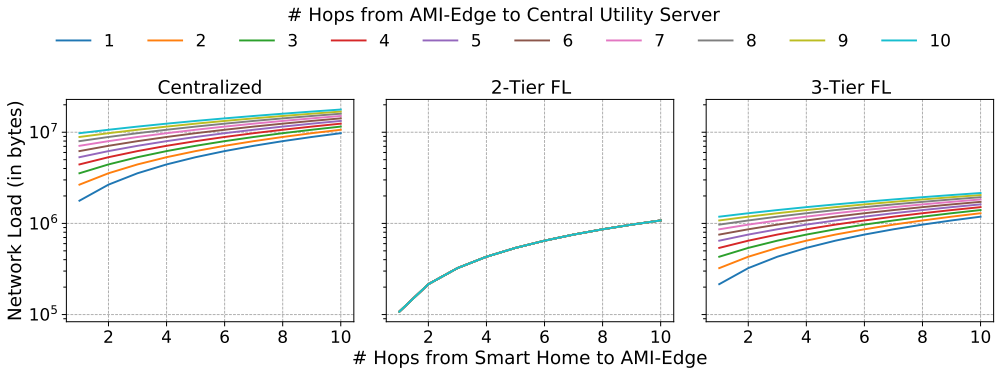

Figure 4.2: The network load for learning using centralized training, 2-tier FL, and 3-tier FL with Eqs. (4.1), (4.2), (4.3) (respectively). We consider the fraction of participating homes for 2-tier and 3-tier FL to be $C = 1$. Number of hops from smart homes to AMI-Edge nodes and AMI-Edge nodes to the central utility server uniformly range from $[1, 10]$.

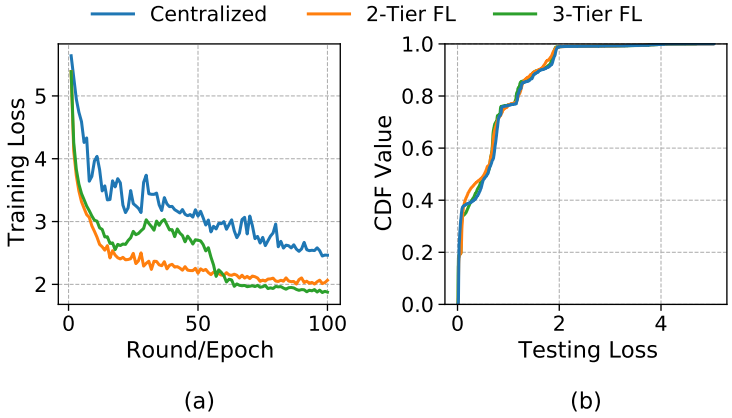

Figure 4.3: Loss performance: (a) Comparison between the training loss over epochs (Centralized) and federated aggregation rounds (2-tier and 3-tier FL). (b) The CDF distribution of the testing loss after the models have been trained — which shows a tight similarity in performance.

collected from a considered subset of the Pecan Street Data, corresponding with a weeks-worth of data, to evaluate the size of the data and parameters, which leads to $|\mathcal{P}_k| \approx 16.8$ *million* bytes being the size (in bytes) of the raw training data hosted on each smart home k. Meanwhile, the size of the considered model parameters is $|\text{model}| = 4896$ bytes. For the FL scenarios (both 2-tier and 3-tier), we consider $C = 1$ (i.e., all smart homes submit locally trained model parameters) to show the maximum possible network load under both scenarios. With the setup established, we now show the distribution of the network loads under each scenario in Fig. 4.2. We observe that both 2- and 3-tier FL notably outperform centralized training in terms of network load/communication cost. This is due to the sheer size of the training data, which vastly exceeds the size of the model parameters.

Also, in 2-tier FL, since communication for training only occurs between smart homes and their respective AMI-Edge nodes, the number of hops between the AMI-Edge nodes and the central utility server has no effect — thus, in Fig. 4.2, the 2-tier FL network load appears to have a single line. Finally, we see that centralized learning is the most costly in terms of network load. However, it is necessary to mention that this analysis is sensitive to the frequency of data communication. For instance, if model parameters are communicated more frequently than raw data, this will affect the communication benefit of the FL approaches. However, given that metering data is dynamic and changes over time, it is likely insufficient to train a model on static data. This would elicit centralized training to collect new data. This trade-off requires further investigation, but these results indicate that there can exist a substantial benefit with respect to network load when employing FL approaches for AMI.

4.6.2 Trained Model Performance

Here, we compare the loss provided by our considered GRU model (discussed in §4.4.1) when trained under centralized, 2-tier FL, and 3-tier FL training approaches. First, we overview the considered hyperparameters for training. For training, we use the standard *mean absolute error* loss function. All scenarios utilize the state-of-the-art Adam algorithm [144] with a learning rate $\eta = 0.001$. Training is done with minibatch sizes of $B = 32$. Centralized training uses 100 training epochs. Meanwhile, both considered FL approaches use 100 federated aggregation rounds with each smart home performing 3 local epochs. Due to the stationary nature of smart homes, we assume that they are always connected to the system infrastructure and thus we consider $C = 1$ — meaning all smart homes participate in local training in each federated aggregation round. Finally, for 2-tier FL, since there is no true global model, we aggregate the training/testing loss and the predicted energy signals provided by all the models aggregated by the 3 considered AMI-Edge nodes.

In Fig. 4.3(a), we see the training losses between the three approaches. On first glance, it would appear that centralized training performs the worst. However, this cannot be claimed because it is being trained directly on the entire data-set — meaning there is more room for error. The FL approaches, on the other hand, are being distributedly trained on smaller shards of the data. However, in Fig. 4.3(b), we compare the testing losses between the models using a set of the Pecan Street data that they were *not* trained on. From this, we find that the models are able to perform comparably under evaluation. Given this comparable performance with respect to test loss and the ameliorated

network load (according to Fig. 4.2), FL shows a considerable benefit for critical systems where resources must be well spent.

4.7 Conclusions & Future Directions

In this chapter, we described the ways in which edge computing and edge intelligence technologies can be leveraged to support and supplement many of the current and future functions in smart distribution grids and particularly AMI. Examples of such functions include wide area monitoring, situational awareness, load monitoring and forecasting, dynamic pricing, and more. To demonstrate the efficiency of edge intelligence, we applied federated learning approach to the task of non-intrusive load monitoring in AMI using 2-tier and 3-tier approaches. In addition to immediate privacy benefits associated with federated learning, our analysis shows that there is a reduction in network load of the system with comparable loss to the centralized learning approaches. There is still much room for further exploration on how the edge and smart distribution grids can operate in tandem. For instance, the data considered in this chapter is fairly limited geospatially, which limits the degree at which the region itself affects the metering data distribution. As mentioned, smart homes in more affluent regions likely consume more energy when compared to smart homes from less affluent areas. This could present a challenge for standard 2-tier FL because the aggregation is more local to the neighborhood covered by that AMI-Edge.

Looking towards the future, there are still open problems and questions to be explored. The trade-off between communication cost and network load regarding these learning methods in smart distribution grids is of great interest. This calls into question how frequently new data comes in to train such models. Additionally, the question of the optimal frequency of performing federated aggregation in this architecture is also worth investigating. Additionally, how we deploy/place edge resources to benefit smart distribution grids in a cost-effective manner to support applications and services of interest also must be explored.

5

Smart Edge-Enabled Traffic Lights Using Federated Reinforcement Learning for Reward-Communication Trade-Off

©2022 IEEE. Reprinted with permission from Nathaniel Hudson, "Smart Edge-Enabled Traffic Light Control: Improving Reward-Communication Trade-offs with Federated Reinforcement Learning," to appear in the proceedings of the 2022 IEEE International Conference on Smart Computing (SMARTCOMP), June 2022.

Traffic congestion is a costly phenomenon of everyday life. Reinforcement Learning (RL) is a promising solution due to its applicability to solving complex decision-making problems in highly dynamic environments. To train smart traffic lights using RL, large amounts of data is required. Recent RL-based approaches consider training to occur on some nearby server or a remote cloud server. However, this requires that traffic lights all communicate their raw data to some central location. For large road systems, communication cost can be impractical, particularly if traffic lights collect heavy data (e.g., video, LIDAR). As such, this work pushes training to the traffic lights directly to reduce communication cost. However, completely independent learning can reduce the performance of trained models. As such, this work considers the recent advent of Federated Reinforcement Learning (FedRL) for edge-enabled traffic lights so they can learn from each other's experience by periodically aggregating locally-learned policy network parameters rather than share raw data, hence keeping communication costs low. To do this, we propose the SEAL framework which uses an intersection-agnostic representation to support FedRL across traffic lights controlling

heterogeneous intersection types. We then evaluate our FedRL approach against Centralized and Decentralized RL strategies. We compare the reward-communication trade-offs of these strategies. Our results show that FedRL is able to reduce the communication costs associated with Centralized training by 36.24%; while only seeing a 2.11% decrease in average reward (i.e., decreased traffic congestion) in our experiments.

5.1 Introduction

According to recent transportation analytics data by INRIX, traffic congestion cost the United States economy $88 billion in 2019 alone [145]. Traffic congestion poses a constant threat to the economy and safety within an urban environment, which can be alleviated by using the compute and communication resources available in smart cities. Urban traffic networks exemplify a typical CPS where data, communication, and connected infrastructure can now jointly optimize traffic operations within a road network. Communication capabilities of the vehicles, traffic lights, and other road-side units (RSUs) powered by vehicle-to-everything (V2X) and vehicular ad-hoc networks (VANETs) provide opportunities for novel strategies to mitigate traffic congestion over large and complex urban road networks [146, 147]. Such strategies may require reliable computing resources for the strict needs of urban traffic networks. The recent advent of *Edge Computing* (EC) [33] pushes compute resources to the network edge via compute node servers, known as "edge servers", that are close to the smart city infrastructure. EC can be used to support more compute-intensive tasks for vehicular networks.

Many recent works trying to support smart decision making for traffic lights (commonly referred to as adaptive traffic signal control) consider *Reinforcement Learning* (RL)-based approaches [148, 149, 150, 151, 152, 153, 154]. RL is a popular technique for training sequential decision-making policies for problems that are highly dynamic and complex. Smart traffic light strategies that incorporate RL typically employ either a centralized [155, 156, 157] or decentralized [158, 153, 159] technique for training policies. In the centralized case, a policy is trained (typically on a roadside server) from the observations collected by detectors and other infrastructural components throughout the system. This central, roadside server then communicates actions to each of the traffic lights. Because the policy is learning over observations throughout the road network, these approaches perform well in terms of maximizing total reward. However, in practice, the amount of communication needed to send all observational data to the server can be costly. Decentralized approaches

push the policy training to the traffic lights based on observations local to that traffic light, meaning less communication is needed since training is local to the traffic light itself. However, in decentralized approaches, the performance of the trained policies can be compromised because policies are learning in an isolated and independent manner. Therein lies a natural trade-off between policy performance with respect to maximizing reward and the communication cost associated with training. To the best of our knowledge, this trade-off has not been formally studied for smart traffic light control with RL.

To this end, we study the reward-communication trade-off for training smart traffic light control policies in an edge-enabled traffic system. We do this by proposing a *Federated Reinforcement Learning* (FedRL) technique inspired by the recent *Federated Learning* (FL) paradigm [160, 161]. Under our FedRL technique, we train traffic lights in a decentralized manner to reduce overall communication costs. Periodically, traffic lights will communicate their current policy network to a roadside edge server (hereafter referred to as "edge-RSU") which will then aggregate the policy network parameters using a weighted averaging method based on total reward. This newly-averaged policy network is then distributed to traffic lights for further training until the next aggregation phase. This aggregation will allow traffic lights to learn from each other without sharing raw observational data. For our FedRL to work, representation of current traffic conditions must be consistent across the road network, even in the face of heterogeneous intersection types. In this way, the representation needs to be *transferable* across road networks and intersections. For this, we design a novel, intersection-agnostic *Markov Decision Process* (MDP) [162] which we refer to as ***S**mart **E**dge-enabled tr**A**ffic **L**ights* (SEAL). The **central contributions** of this work can be summarized as follows:

- Design a novel, intersection-agnostic MDP for representing traffic conditions at traffic lights which we call SEAL. SEAL is designed to have a general representation of traffic conditions at intersections.
- Proposal of a *Federated Reinforcement Learning* (FedRL) approach for training RL decision-making policies for smart traffic light control.
- Improve reward-communication cost trade-off associated with solving SEAL using our proposed FedRL approach by reducing communication costs up to 36.24% on average while losing 2.11% on average when compared to Centralized training.

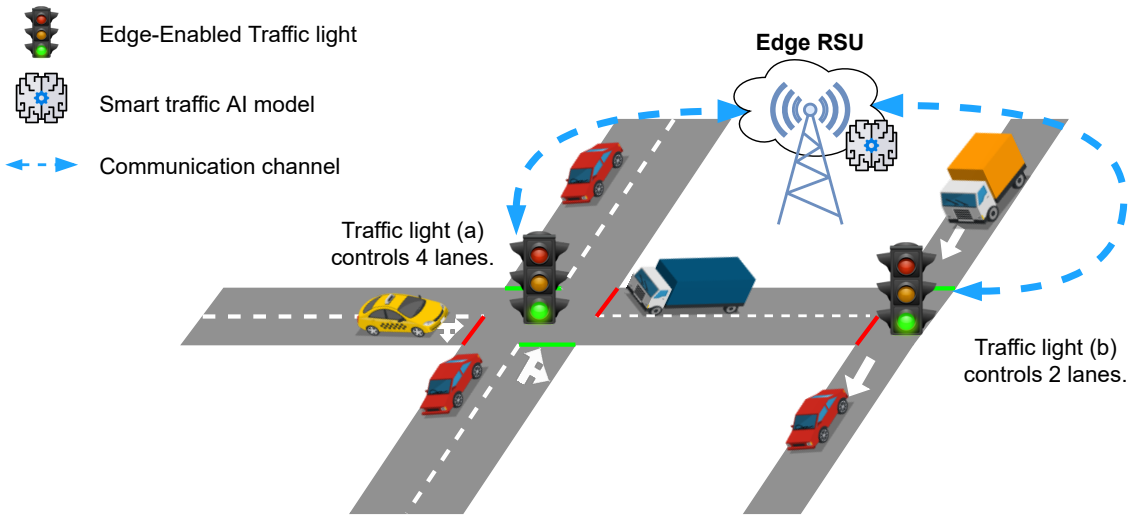

Figure 5.1: Example of our traffic system where traffic lights communicate with an edge-enabled roadside unit (Edge-RSU).

5.2 System Description

We now describe the system requirements for traffic infrastructure, data, and communication capabilities for our model. Fig. 5.1 shows a typical traffic environment where our model could be deployed. Our system considers a road network with one or more intersections (depending on the road topology), each equipped with a traffic light $k \in \mathcal{K}$ where $\mathcal{K}$ denotes the set of traffic lights in the entire system. Each traffic light $k \in \mathcal{K}$ controls the traffic flow entering the intersection through an incoming lane. A set of such *controlled lanes* is denoted by $\mathcal{L}_k$. A traffic controller, either located at each intersection or at a server calculates a "phase state" φ_k^t for the traffic lights at a given traffic light k at time-step t. The assigned phase state is such that a traffic light will be assigned a green, yellow or red "signal state", represented by `G`,`y`,`r`, respectively. Therefore, a phase state is a string representing the signal states of the traffic lights at all controlled lanes at an intersection. For example, the phase state for an intersection with eight controlled lanes would be `GyrrGyrr`. For a visual example of phase states, refer to Fig. 5.2. Note that the phase states are assigned such that the vehicles with conflicting traffic flows are not allowed to access the intersection at once. The length of the phase state is based on the number of incoming controlled lanes at a given intersection. Our model also expects that the vehicles obey the traffic regulations and do not violate the assigned phase permissions indicated by the traffic lights. Finally, in our system, we assume each traffic light $k \in \mathcal{K}$ cannot change phase states until 4 seconds have elapsed since k's last phase state change. Further, we assume each traffic light $k \in \mathcal{K}$ *must* change after 120 seconds have elapsed since k's last phase

state change. These timings are calculated in accordance with the U.S. federal highway administration (FHWA) guidelines based on average traffic behavior [163] and can be changed as per traffic regulatory requirements. This is enforced for all training and evaluation.

Traffic infrastructure is equipped either with road-side sensors installed within every controlled lane to measure traffic parameters such as lane occupancy, average traffic flow speed, etc. (detailed in §5.3) or have connected vehicles to report such data to the traffic lights by utilizing the connected infrastructure. Traffic lights are equipped with edge compute resources to process the data and perform local learning. The edge resources also enable the connectivity among all traffic lights within the traffic network as well as the centralized cloud server to enable global optimization of the learning models. For simplicity, we assume the presence of a *single* deployed *edge-RSU* server in the region that maintains communication channels to all the traffic lights in a given region to support additional processes. Additionally, traffic lights are also equipped with compute resources as well as the edge-RSU server. As a simplifying assumption, we assume compute resources at both the traffic lights and the edge-RSU server are sufficient to train policies for smart traffic decisions. Succinctly, this work aims to improve the implicit reward-communication trade-off associated with distributed learning solutions to support smart traffic systems using FedRL.

5.3 Proposed SEAL Model Definition

Here, we define the ***S**mart **E**dge-enabled tr**A**ffic **L**ights* (SEAL) system. SEAL is modeled as a *Markov Decision Process* (MDP) [162] with the goal to minimize traffic congestion in road networks. SEAL's novelty is in defining a general state space representation that can describe current traffic conditions at a traffic light in an intersection-agnostic way. This is necessary to support policy aggregation in our FedRL approach (discussed later in §5.4.3).

The work most similar to ours is that of Zhou et al.'s DRLE framework in [158]. This work is able to consider a distributed multi-agent RL approach to smart traffic light control with convergence guarantees. However, this does not consider the possibility of traffic lights themselves training their own policy networks. Instead, the DRLE framework sets traffic lights to communicate their local state observations to a roadside server to perform state aggregation to form a "global" state. This global statefulness allows for convergence guarantees, but may not be attractive for future solutions where traffic lights may collect large volumes of data (e.g., hyper-spectral images, videos, LIDAR imaging, etc.) to make decisions. Having large amounts of traffic lights stream these data

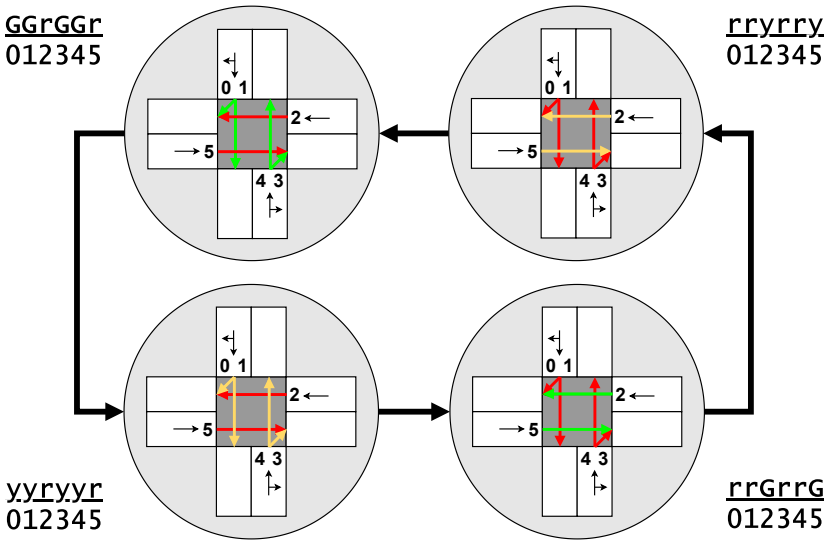

Figure 5.2: Example traffic light action transition graph. Consider the given traffic light k's current phase state is `GGrGGR`. If the action $a_k^t = 1$ at time-step t, then the phase state for k will transition to `yyryyr` if sufficient time has elapsed since its last transition. Otherwise, its phase state remains the same, unless too much time has elapsed since its last change.

in real-time to make timely decisions may not scale well. Thus, we consider SEAL. Future works investigate possible convergence bounds of SEAL is of interest but is beyond the scope of this work.

5.3.1 Action Space

In prior works investigating the use of RL for traffic light control, various kinds of actions have been considered. These include phase switch [158, 159], phase duration [152], and the phase state itself [150]. The phase state considers a discrete space of size n where n is the number of possible states for a traffic light. Since phase state depends on the number of controlled lanes and hence the traffic lights at an intersection, it is infeasible to aggregate knowledge among the intersections with varying topologies. For this work, we consider a simpler phase switch approach in which we consider each traffic light $k \in \mathcal{K}$ in time-step t to take an action $\mathbf{a}_k^t \in \{0,1\}$ where $\mathbf{a}_k^t = 1$ signifies that traffic light k will attempt to change to the next phase state. Otherwise, $\mathbf{a}_k^t = 0$ signifies no phase state change will be attempted by traffic light k at time-step t.

Note, if a traffic light k attempts to change in some time-step t (i.e., $\mathbf{a}_k^t = 1$), a change can only occur if enough time has elapsed since its last change; further, a traffic light k will be forced to change its phase state regardless of its action if *too* much time has elapsed since its last change. This is due to the phase state timer (discussed in §5.2) to ensure policies mean mandatory regulations

related to road safety [163]. Refer to Fig. 5.2 for an illustrated example of phase state logic and transitions made when $\mathbf{a}_k^t = 1$.

5.3.2 State Space

State space features consist of the following for a traffic light k in time-step t: *lane occupancy* (o_k^t), *halted lane occupancy* (h_k^t), *average speed* (ψ_k^t), and *phase state ratios* ($\varphi_k^t(\cdot)$) for all possible phase states (e.g., green, yellow, red).

Lane Occupancy

The average ratio of occupancy across all lanes controlled by a traffic light k in time-step t. Each traffic light k controls some set of lanes. Thus, we consider the occupancy of a lane l to be how much of a lane's length (in meters) is occupied by vehicles (as a ratio). However, we average this across all lanes controlled by traffic light k. The formal definition for lane occupancy is provided below:

$$o_k^t \triangleq \frac{\sum_{l \in \mathcal{L}_k} \sum_{v \in \mathcal{V}_l^t} len(v)}{\sum_{l \in \mathcal{L}_k} len(l)} \tag{5.1}$$

where $\mathcal{L}_k$ is the set of lanes controlled by traffic light k, $\mathcal{V}_l^t$ is the set of vehicles occupying lane l in time-step t, and $len(\cdot)$ is the length of the vehicle or lane (in meters).

Halted Lane Occupancy

SEAL's goal is to minimize congestion in road systems. Thus, we consider how much of a lane is occupied with halted vehicles. As such, we consider h_k^t to be the *halted lane occupancy* of traffic light k in time-step t when we consider a vehicle to be halted if its current speed is ≤ 0.1 meters/second. Thus, we define h_k^t below:

$$h_k^t \triangleq \frac{\sum_{l \in \mathcal{L}_k} \sum_{v \in \mathcal{H}_l^t} len(v)}{\sum_{l \in \mathcal{L}_k} len(l)} \tag{5.2}$$

where $\mathcal{H}_l^t$ is the set of halted vehicles occupying lane l in time-step t.

Average Speed

We also consider the average speed (ψ_k^t) among vehicles occupying lanes controlled by a traffic light k at time-step t as a feature. Similar to the other features, this one is also normalized as a ratio in the range $[0,1]$. The formal definition is below:

$$\psi_k^t \triangleq \begin{cases} \frac{\sum_{l\in\mathcal{L}_k}\sum_{v\in\mathcal{V}_l^t}\min\left(spd_v^t, spd_l^{\max}\right)}{\sum_{l\in\mathcal{L}_k}\sum_{v\in\mathcal{V}_l^t} spd_l^{\max}} & \left|\bigcup_{l\in\mathcal{L}_k}\mathcal{V}_l^t\right| \geq 1 \\ 1.0 & \text{otherwise} \end{cases} \tag{5.3}$$

where spd_v^t is the moving speed of vehicle v in time-step t and $spd_l^{\max}$ is the speed limit on lane l. The second case in Eq. (5.3) is for cases when there are no vehicles occupying lanes controlled by traffic light k.

Phase State Ratio

The current phase state of a traffic light has been used as feature in prior works (namely, [158]). This is possible because simple road networks are considered with homogeneous intersections where traffic lights have the same sets of possible phase states. To handle *heterogeneous phase state sets* across different intersection types, we instead represent the *ratio* of how each possible traffic light signal (e.g., green, yellow, red) makes up the entire phase state. Thus, we denote the ratio of a traffic light signal for a traffic light k in time-step t by $\varphi_k^k(\cdot) \in [0,1]$. For instance, given a phase state at traffic light k in time-step t `GGrGGr`, we denote how much of the phase state are red lights, `r`, by $\varphi_k^t(r) = \frac{2}{6}$ (similarly for prioritized green lights, `G`, $\varphi_k^t(G) = \frac{4}{6}$). Because we represent the *ratio* rather than assign an arbitrary discrete value to represent the entire phase state, the representation is general and can be used across different road networks with various intersections. It should be noted that $\sum_{p\in\mathcal{P}_k}\varphi_k^t(p) = 1\ (\forall k,t)$ where $\mathcal{P}_k$ is the set of phase states for traffic light k.

5.3.3 Reward Function

The goal of SEAL is to reduce congestion in a given road network. With that in mind, we let reward r_k^t for a traffic light k at time-step t be a function of both *lane occupancy* (o_k^t) and *halted lane occupancy* (h_k^t). We define it below:

$$r_k^t \triangleq -(o_k^t + h_k^t)^2. \tag{5.4}$$

These state space features are summed to penalize traffic lights with more congestion. We let halted vehicles to incur more penalty since they contribute to both lane occupancy and halted lane occupancy. From there, we define the *total reward*, r^t, over the road whole network at time-step t as

$$r^t \triangleq \sum_{k \in \mathcal{K}} r_k^t. \tag{5.5}$$

5.3.4 Communication Model

As discussed in §5.2, we require robust communication capabilities between vehicles, traffic lights and edge-enabled RSUs to support smart traffic control. Depending on the training approach (detailed in §5.4), a traffic control system must account for different communication channel utilization and their incurred costs. We therefore consider the following 6 different types of possible communications that can take place under the SEAL system: *(i)* policy network parameters from edge-RSU to traffic light, *(ii)* policy network parameters from traffic light to edge-RSU, *(iii)* action from edge-RSU to traffic light, *(iv)* observations from traffic light to edge-RSU, and *(v)* vehicle-to-infrastructure (V2I) communication from vehicle to traffic light. We will evaluate the associated communication costs while training of our proposed model in §5.6. To reiterate, we assume that edge-enabled traffic lights and the edge-RSU have sufficient compute capacity to performing policy training. Thus, we do not consider compute constraints and focus on communication cost instead.

5.4 Training Algorithms

The goal of SEAL is to learn optimal traffic light control policies to minimize congestion for a given road network. To solve the SEAL model, we adopt model-free reinforcement learning techniques. More specifically, we will incorporate the recent *Proximal Policy Optimization* (PPO) [164] algorithm. Solutions to SEAL will aim to find a smart traffic light control policy, π, such that

$$Q^{\pi}(\mathbf{s}, \mathbf{a}) = (1-\gamma) \cdot \mathbb{E}\left[\sum_{t=1}^{\infty} (\gamma)^{t-1} \cdot r^t | \mathbf{s}^1 = \mathbf{s}, \mathbf{a}^1 = \mathbf{a}\right] \tag{5.6}$$

is maximized where the policy is a decision-making function $\pi : S \mapsto A$ and γ is the discount factor. Eq. (5.6) is known as the Q-function. The optimal policy that maximizes the Q-function is defined as $\pi^* = \arg\max_{\pi} Q^{\pi}(\mathbf{s}, \pi(\mathbf{s}))\ \forall s$. For the sake of convenience, we denote $Q(\mathbf{s}, \mathbf{a}) = Q^{\pi^*}(\mathbf{s}, \mathbf{a})$, $\forall (\mathbf{s}, \mathbf{a})$ where $\mathbf{s}$ and $\mathbf{a}$ are a state and action, respectively.

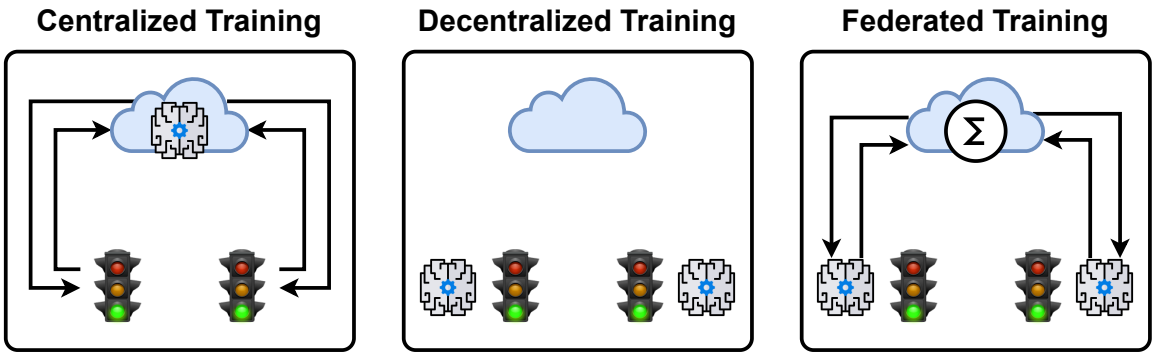

Figure 5.3: Training approaches considered for solving SEAL.

RL algorithms can be implemented in real-world systems in various ways. As such, we consider 3 different approaches for facilitating the PPO algorithm to solve SEAL: *(i) centralized training*, *(ii) decentralized training*, and *(iii) federated training*. A visual example of how these approaches compare can be found in Fig. 5.3. For a comprehensive overview on the theory of RL, please refer to [165].

5.4.1 Centralized Training

Under centralized training, there is a single policy network that is hosted on the nearby edge-RSU. At each time-step t, each traffic light $k \in \mathcal{K}$ submits their current state $\mathbf{s}_k^t$ to the edge-RSU which then returns an action $\mathbf{a}_k^t$ to traffic light k. Since a single policy network is learning across *all* observations in the system, it is expected to learn the optimal policy faster than other approaches. However, this is at the expense of incurring a large amount of overhead in terms of communication cost because of the traffic light having nonstop communication with the edge-RSU to take an action. For this work, we view this approach as an upper bound in terms of most quickly learning the optimal policy, π^*.

Centralized Training Communication Costs

Decision-making in a centralized manner requires traffic lights to always communicate to the edge-RSU leading to higher communications. Under Centralized training, the following communications take place at each time-step: actions from the edge-RSU to traffic lights, observations from traffic lights to edge-RSUs, and V2I communications from vehicles to traffic lights.

5.4.2 Decentralized Training

Unlike centralized training, decentralized training equips each traffic light $k \in \mathcal{K}$ with a policy network that aims to independently learn an optimal local policy for traffic light k, π_k^*, for optimizing reward using only observations local to that traffic light. In essence, if all traffic lights in the system are able to learn an optimal policy, then that can benefit the entire road network. Zhou et al. in [158] proved that a decentralized training approach using per-traffic light policies for smart traffic light control, can converge to a centralized approach if given infinite time. In general, this approach can attain good performance if given enough time. While the decentralized approach is bested by the centralized approach in finding an optimal policy, since the latter is learning from global observations, the former approach is of interest as it requires less communication.

Decentralized Training Communication Costs

In the decentralized case, since the traffic lights never communicate to the edge-RSU for making decisions, little communication occurs. The only communication that takes place is V21 communication from vehicles to traffic lights.

5.4.3 Federated Training

With the expectation that decentralized training will not perform as well as centralized training due to policies learning over fewer observations, but will require less communication, we wish to achieve the best of both worlds. A novel contribution of this work is that we leverage the findings of the recent *federated learning* (FL) paradigm [160, 161] for distributed systems. Here we apply it to decentralized training to allow the traffic lights to learn from each other without needing to communicate raw data. We refer to this notion aptly as *Federated Reinforcement Learning* (FedRL) [166, 167]. FL has shown to reduce communication cost in the literature [168] while providing an immediate layer of privacy because no raw data are communicated. These are crucial advantages for smart traffic light control for future systems. For instance, consider a system that considers live video feed as a feature in the state space representation. Because identifying information (e.g., license plate numbers and faces of pedestrians) may be included, privacy is crucial. Additionally, such data may be very large and incur hefty data transmission costs. As such, we will focus on the benefit of federated training for smart traffic light control with respect to the trade-off between communication cost on the system and maximizing reward.

In FedRL, the traffic light agents training their own policy networks will periodically communicate the learned policy network parameters to the edge-RSU. The edge-RSU will then aggregate them using an averaging function. The newly aggregated policy network parameters are then communicated back to the traffic lights for further learning. Aggregation will occur after a number of time-steps occurs. We refer to this time period as a frame and denote it by F. We denote the policy network parameters learned by traffic light k at the end of frame F by ω_k^F.

In [161], the *federated averaging* (FedAvg) technique was proposed. This technique addresses the challenge of non-independent and identically distributed (iid) data distributions across different client devices. FedAvg uses a weighted average of the client's locally-updated model parameters based on the number of data items owned by that client. This weight combats non-iid data distributions common in distributed systems. For the sake of this work, we consider a simplifying assumption that traffic lights have identical data sampling rates — resulting in the same amount of observations. Below is the definition of the averaging we consider,

$$\omega^{F+1} \triangleq \sum_{k \in \mathcal{K}} \frac{1}{|\mathcal{K}|} \omega_k^{F+1} \tag{5.7}$$

where the newly-aggregated, global parameters ω^{F+1} is the average of the parameters collected from all the traffic lights. These parameters are then sent back to the traffic lights at the start of frame $F+1$ to resume training. Asynchronous aggregation techniques to address heterogeneous data sampling rates among traffic lights is beyond the scope of this work.

In addition to this aggregation function, we consider a reward-based aggregation technique. However, our experiments showed that it is

$$\omega^{F+1} \triangleq \sum_{k \in \mathcal{K}} \frac{r_k^F}{r_k} \omega_k^{F+1} \tag{5.8}$$

where r_k^F is the total reward received by traffic light k over the training frame F and is the total reward received by all traffic lights during training frame F (i.e., $r_k = \sum_{k \in \mathcal{K}} r_k^F$). Note, if $r_k = 0$, then aggregation is done using Eq. (5.7). Since reward under SEAL is never positive (i.e., $r_k^t \leq 0 \ (\forall k, t)$), traffic lights that achieve the least amount of reward will have the greatest weighting coefficients for aggregation in Eq. (5.8). The intuition for this design is based on findings from works in conventional federated learning. For instance, Cho et al. in [169] analyze convergence in federated learning processes and find that bias towards clients with higher amount of loss (i.e., perform worse) improves convergence. Their intuition is that clients that have high loss have data that can best improve the model. Meanwhile, the global model can be seen as doing a good job

for making predictions for the data hosted on clients with low loss. In our case, we consider traffic lights with good reward to be less important. Intuitively, traffic lights with no vehicles with nearby will receive maximum reward — which is not helpful for learning how to best train traffic lights. However, we find from simple evaluation that this aggregation function was less stable and was often bested by the aggregation function defined in Eq. (5.7). As such, we focus on Eq. (5.7).

Federated Training Communication Costs

With federated training, communications that occur at each time-step are mostly identical to that for decentralized training (discussed in §5.4.2). The only difference is at the end of each frame (which occur less frequently than each time-step), 2 additional communications occur: policy network parameters from edge-RSU to traffic lights and policy network parameters from traffic lights to edge-RSU.

5.5 Experiment Design

We implement the SEAL framework using the Python programming language. Further, we implement the training approaches described in §5.4 using the SUMO traffic simulator [170] for the traffic simulation and Ray's RLlib [171] toolbox for the RL pipeline. Our software serves as the interface for these tools to fit our work's very specific needs. Thus, we only train the policy networks using PPO using simulations with these tools.

5.5.1 Considered Road Network Topologies

For training the policies using Ray's RLlib [171] and performing evaluation via simulation, we consider 3 road network topologies provided in Fig. 5.4: *(a)* Grid-3×3, *(b)* Grid-5×5, and *(c)* Grid-7×7. Roads on the border of the network have 1 lane going each direction, with the number of lanes going north/south and east/west increasing by 1 when approaching the center north/south and east/west roads. This is to introduce heterogeneous road network topologies. For an example, refer to Fig. 5.4. For simplicity, we do not allow vehicles to make turns to prevent the vehicles from getting stuck in the simulation. Note that this is a limitation of SUMO and SEAL's design is general enough to support turning vehicles. Each training approach (discussed in §5.4) will learn policies over each road network topology. Vehicles routes for training and evaluation are randomly generated using

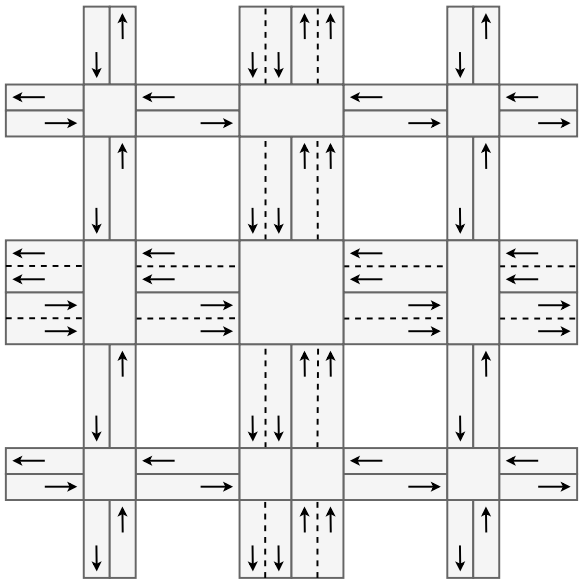

Figure 5.4: Example Grid-3 $\times$ 3 road network with heterogeneous intersection types. Note that the number of lanes increase as roads are more central.

the `randomTrips.py` module provided by SUMO with 360 *vehicles per lane per hour* (VPLPH) generated. According to SUMO's documentation, vehicles are inserted into the network using any of the edges in the system following a binomial distribution. Further, SUMO's documentation notes that for large road networks this distribution will approximate a Poisson distribution. Analysis of how road traffic distribution affects learning is left to future work.

5.5.2 Training Parameters

We use *Proximal Policy Optimization* (PPO) [164] to train policies to solve SEAL. We use the following hyper-parameters. The learning rate is 5×10^{-5}. SGD minibatch size is 128. PPO *CLIP* parameter is set to 0.3. Target value for KL divergence is 0.3. Train batch size is 4000 time-steps. (Note policy network parameter aggregation, described in §5.4.3, occurs every 4000 steps.) Roll-out fragment length (size of batches collected from each worker) is 200. We use *Generalized Advantage Estimator* (GAE) and the GAE parameter is set to 1.0. The VF clip parameter is set to 10.

5.6 Results & Discussion

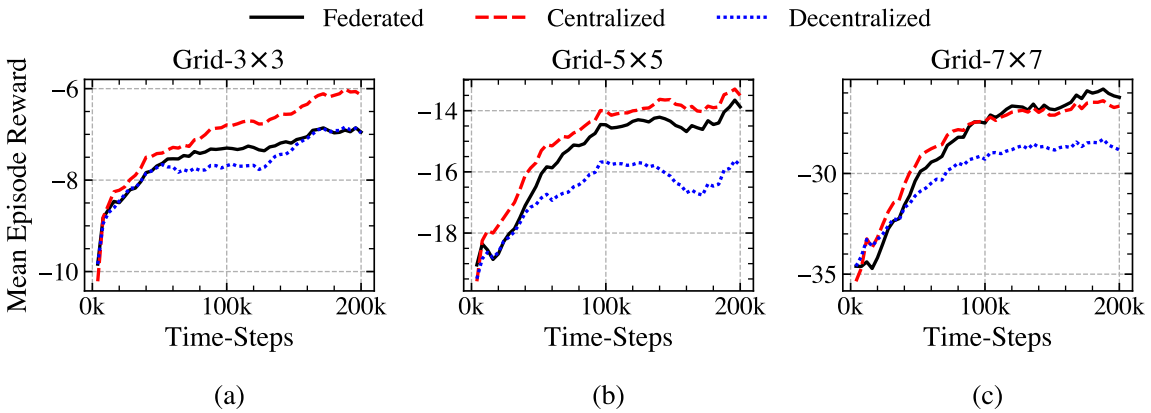

Figure 5.5: Learning curves with each training approach on each road network.

5.6.1 Reward Evaluation During Training

First, we compare the different training strategies discussed in §5.4 in terms of the reward achieved by the policy networks during training. In Fig. 5.5, we can see the learning curves of each training strategy when used on each of the 3 road network topologies described in §5.5.1. From these results, we find that, in general, make the following observations: *(i)* Centralized training generally achieves the greatest reward, *(ii)* Decentralized training generally achieves the worst reward and, *(iii)* Federated training achieves greater reward than Decentralized training (and often nearly matches that of Centralized training). These observations are fairly intuitive. Since Centralized training trains a single policy network over all observations collected in the environment, it has more to learn from. Conversely, with Decentralized training, each traffic light learns independently using its own observations — meaning each traffic light's policy learns over fewer observations. Since Federated training expands on Decentralized training by allowing parameter aggregation among the policy networks learned by the traffic lights, the traffic lights are essentially able to learn from each other without explicitly sharing observations and other raw data. More specifically, we find that Decentralized training suffers from an 8.01% drop in reward compared to the Centralized training. Meanwhile, Federated training only suffers from an 2.11% drop in reward compared to Centralized training.

5.6.2 Communication Cost Evaluation During Training

Given Federated training is able to more closely approximate the reward achieved by Centralized training when compared to Decentralized training, we next compare the communication costs as-

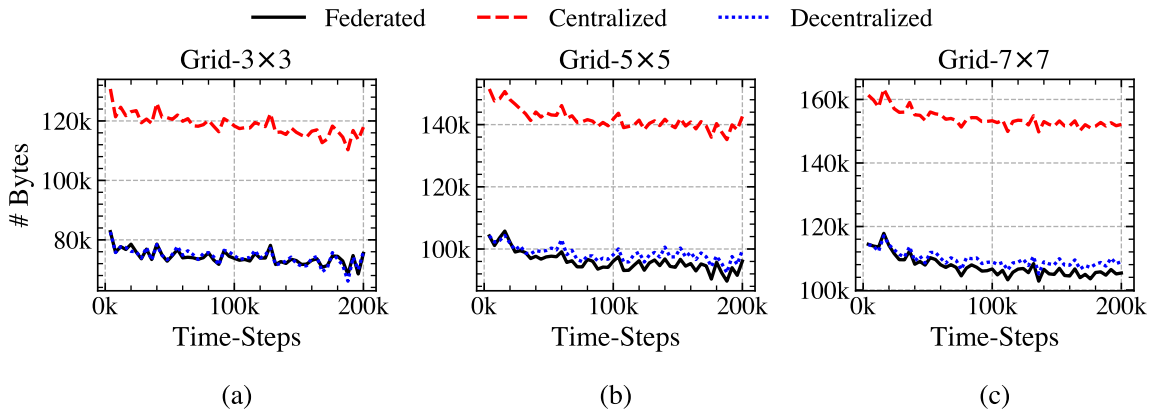

Figure 5.6: Communication cost (i.e., data size in bytes) transmitted during training time under each training strategy for each road network.

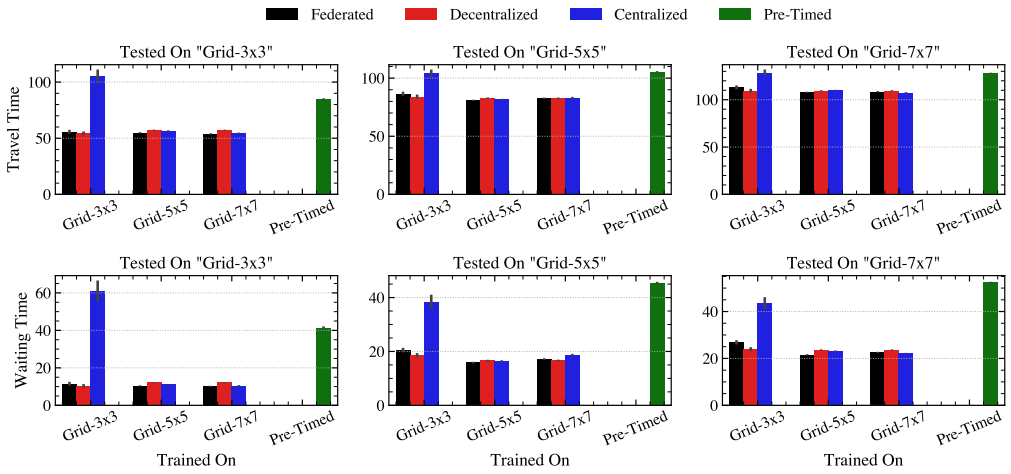

Figure 5.7: Evaluation of trained policy networks on each road network using trip metrics, namely Travel Time (top row) and Waiting Time (bottom row). We compare the results to a Pre-Timed phase transition model as a baseline. Results confirm the RL-based solutions generally outperform the baseline.

sociated with each training strategy. We do this by tracking the number of communication that occurs (refer to §5.3.4) and the number of times each communication type occurs by the amount of bytes needed to transmit the data for that communication. In Fig. 5.6, we compare the size of the data needed to be communicated through the system during training using each of the training strategies under each of the road network topologies. There is a glaring difference in terms of communication efficiency between Centralized and Decentralized/Federated. Because Centralized training requires constant communication between the edge-RSU and the traffic lights in order to transmit observations, actions, and other data, it naturally incurs much greater communication cost.

Meanwhile, Decentralized and Federated training greatly reduce this cost due to them keeping communication mostly between the vehicles and the traffic light. The only communication between the Edge-RSU and the traffic lights under Federated training is when policy network parameters are aggregated after each frame concludes. It is interesting to note that Federated is able to best Decentralized training in terms of communication cost in these results. This is due to the Federated training strategy producing better policy networks and removing vehicles from the system more efficiently than the Decentralized model — resulting in less vehicle-to-infrastructure communication. More numerically speaking, from our results Decentralized and Federated training are able to achieve a communication cost reduction of 34.65% and 36.24%, respectively, when compared to Centralized training.

5.6.3 Trained Policy Network Performance

Here, we are interested in *two* questions: (1) Can RL-based traffic lights trained with SEAL improve traffic conditions? (2) Can policy networks trained with SEAL perform well when used on road networks they were not trained on? To answer the first question, we compare our trained policy networks against a standard traffic light control baseline: a *pre-timed control* [163] where traffic lights cycle through phase states at fixed time intervals. We support this comparison using real-world traffic metrics to evaluate the experience of drivers in the system. Namely, we consider both "Travel Time" and "Waiting Time". The former is the total amount of (simulation) time taken for vehicles to reach their destination; the latter is the amount of (simulation) time vehicles are waiting to move at a traffic light. The results of this evaluation are shown in Fig. 5.7. We see that in nearly all cases, the RL-based training strategies outperform that of the Timed-Phase baseline. The only outlier is the Centralized trainer when learning in the Grid-3 $\times$ 3 road network. As for the second question regarding possible transferability of the policy networks, we observe in Fig. 5.7 that the policy networks are generally able to perform comparable to one another (ignoring the Centralized trainer when trained on Grid-3 $\times$ 3). This generally holds true for policy networks being tested on the same road network they were trained on when compared to policy networks trained on other networks. These results serve to motivate the use of RL-based approaches for future smart traffic applications. We find that (on average) Centralized, Decentralized, and Federated reduce *travel time* compared to Pre-Timed by 11.63%, 18.16%, and 18.14%, respectively. Also, we find that (on average) Centralized, Decentralized, and Federated reduce *waiting time* compared to Pre-Timed by

42.81%, 58.92%, and 58.93%, respectively. The underperformance of Centralized here, compared to Decentralized and Federated, is likely due to the outlier scenarios when its trains on Grid-3×3. We attribute these anamolies to potential overfitting, though further experiments are needed.

5.7 Related Works

Improving traffic light signal control in road networks has been a widely studied subject. Much work is being done to improve traffic conditions by developing *adaptive traffic signal control* (ATSC) where traffic lights adapt intelligently based on current traffic demands [172]. Many different techniques have been considered for realizing ATSC. Early works considered linear optimization frameworks [173].While linear programming is straightforward, it is not an appropriate match for ATSC because of the highly dynamic nature of real-world traffic systems — making accurate objective functions and constraints difficult to define. Genetic (or evolutionary) algorithms have also been considered in prior works [174]. In the early 2000s, initial works focusing on the application of *Reinforcement Learning* (RL) techniques for ATSC were published [149, 155]. While seminal, these initial works considered very simple road network scenarios. With advancements in both vehicular communication [146, 147] and RL algorithms [165], interest in RL for ATSC (or smart traffic) has been renewed. However, recent RL algorithms use more complex policy networks that require more compute resources to train.

Works considering RL for smart traffic light signal control have greatly increased over the years [150, 152, 153]. Because of the large number of entities in a traffic system (e.g., multiple traffic lights, multiple vehicles), *multi-agent* RL techniques have been applied to smart traffic light control [159, 154]. El-Tantawy et al. in [159] propose a multi-agent RL framework where agents can either be independent or collaborative in how they make decisions with other traffic light agents. Chu et al. in [151] propose a decentralized, multi-agent RL framework to provide robust learning with using a scalable framework. Chen et al. in [148] propose a decentralized actor-critic model and a difference reward method to accelerate the convergence of the trained policies for smart traffic light control. Mousavi et al. in [156] study both, policy- and value-based deep reinforcement learning approaches for smart traffic light control. However, they only consider a single intersection, where the state space is a screenshot of the intersection provided by a traffic simulator. These works focus on improving training first and foremost. For instance, the communication cost for training these policies is neglected.

Edge Computing (EC) [33, 17] is a recent enabling technology that pushes compute resources to the network edge. This has become an increasingly popular context for deploying AI (e.g., machine learning, deep learning, and RL) services to the network edge to provide low-latency intelligence. A significant recent work by Zhou et al. in [158] studied the applicability of edge computing for decentralized RL for smart traffic lights. A central contribution of this work is the theoretical guarantees that show that their proposed decentralized framework can provide a near-optimal guarantee on reduced traffic if given enough time. Different from this work, we design a framework that allows heterogeneous traffic lights to train policy networks in a federated manner to reduce communication costs.

The *central gap* in the literature related to RL for smart traffic light control is that the trade-off between reward and communication cost has been neglected. Additionally, recent advancements in the realm of *Federated Learning* (FL) or, more specifically, *Federated Reinforcement Learning* (FedRL) has yet to be applied to the smart traffic control problem.

5.8 Conclusions

In closing, this work to the best of our knowledge, is the first to approach smart traffic light control using *Federated Reinforcement Learning* (FedRL) in an edge computing-enabled system. We do this by proposing SEAL, which is an intersection-agnostic Markov Decision Problem for smart traffic light control to support aggregating learned policy network parameters across heterogeneous intersection types. This allows traffic lights to learn from each other's experiences without sharing raw experience data which reduces communication workloads (while providing some level of privacy). Our experiments demonstrate that SEAL combined with FedRL approach is able to closely match the rewards provided by a Centralized training approach (only a 2.11% decrease) when compared to the Decentralized approach that shows a 8.01% drop in reward. Further, our FedRL approach reduces the communication cost by 36.24% when compared to Centralized training. Hence, FedRL improves the implicit reward-communication trade-off for distributedly training smart traffic systems. In the future, we aim to extend our work to further analyze the theoretical bounds of SEAL and to study its effectiveness in small robotic testbed systems.

6

Conclusions & Final Remarks

Turning the ambitious vision of smart cities from fantasy into reality is a daunting task. It will require robust information communications and technology infrastructure to achieve. Edge computing is a viable candidate for supporting many of the expected services needed to support smart cities. Further, pushing artificial intelligence-based solutions to the edge (i.e., edge intelligence) can aid in transforming data collected throughout smart cities into knowledge while also automating smart decision-making in a resource-conscious way.

6.1 Findings

In Chapter 2 we studied the problem of placing and scheduling edge intelligence services. We first cast the problem as an integer linear program and proved that it is, in fact, NP-*hard* to solve. Next, we proved that scheduling can be solved optimally via a greedy algorithm given a set of placement decisions. We then proved that a greedy algorithm for placement achieves a $(1-1/e)$-approximation of the optimal solution. With this, we propose two greedy algorithms: one that theoretically guarantees this approximation bound and another that is more efficient while empirically matching the performance of the former algorithm.

Chapter 3 builds on top of prior work that showed the relationship between lossy compression techniques and the achieved accuracy of deep learning models for image classification. It was shown this relationship can be fitted with a Gompertz function. Using this finding, this chapter defines a nonlinear problem for joint compression and offloading decisions in a 3-tier edge computing system with the goal to provide high accuracy to end users while minimizing overall communication costs

via lossy compression. Using a block coordinate descent method, we were able to find a close-to-optimal solution. Additionally, we proposed an iterative algorithm that is able to closely match its performance in an efficient manner — besting all other benchmarks.

In Chapter 4 describes how edge computing can aid smart energy distribution grids in the context of smart cities. Further, this chapter focuses on the application of non-intrusive load monitoring for smart energy distribution grids. This problem is approached using 2-tier and 3-tier federated learning to train a deep neural network to disaggregate energy loads. The results showed that using federated learning through the edge provides comparable loss as conventional centralized techniques while greatly reducing overall communication costs.

In Chapter 5 demonstrates the applicability of edge computing infrastructure to support future smart traffic control systems that adapt to current traffic conditions. This is done using deep reinforcement learning and applying averaging techniques found in federated learning (which we refer to as *federated reinforcement learning*). Our results confirm that federated reinforcement learning is also able to reduce the communication costs associated with training models for making smart decisions related to traffic light control in smart road systems.

6.2 Future Work

There still remain countless challenges and problems on the horizon of realizing smart cities in future urban environments. Some of these remaining challenges include expanding upon the work established in this thesis. For instance, considering more dynamic systems for optimization seen in Chapters 2 and 3 is of utmost importance. Given how diverse and large smart cities will be, it is absolutely crucial that more dynamic settings are considered for future works. Additionally, the notion of data freshness is an obvious next step for the work of this thesis. Consider Chapter 4, where we use energy data collected by individual smart homes to train a predictive model in a federated fashion. The question of how long smart homes should hold onto data before discarding it is interesting. Will old data harm predictive accuracy of the model? How do storage constraints at the smart home affect the decision to hold onto data? There are possible trade-offs here that must be further explored. Finally, Chapter 5 only aims to optimize road conditions. However, road traffic has implicit and interdependent relationships with many other smart city systems and processes. For instance, traffic light control could likely benefit from knowing about a protest occurring in the city that was detected via crowd-sensing solutions deployed elsewhere in the system. Modeling smart

traffic as an interdependent process is crucial in order to envision this ultimate, ambitious vision of a smart city where data and technology seamlessly cooperate.

Bibliography

[1] T. Bakıcı, E. Almirall, and J. Wareham, "A smart city initiative: the case of barcelona," *Journal of the knowledge economy*, vol. 4, no. 2, pp. 135–148, 2013.

[2] M. O'grady and G. O'hare, "How smart is your city?," *Science*, vol. 335, no. 6076, pp. 1581–1582, 2012.

[3] F. Li and Y. Wang, "Routing in vehicular ad hoc networks: A survey," *IEEE Vehicular technology magazine*, vol. 2, no. 2, pp. 12–22, 2007.

[4] C. Cooper, D. Franklin, M. Ros, F. Safaei, and M. Abolhasan, "A comparative survey of vanet clustering techniques," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 657–681, 2016.

[5] H. Hasrouny, A. E. Samhat, C. Bassil, and A. Laouiti, "Vanet security challenges and solutions: A survey," *Vehicular Communications*, vol. 7, pp. 7–20, 2017.

[6] M. Gerla, E.-K. Lee, G. Pau, and U. Lee, "Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds," in *2014 IEEE world forum on internet of things (WF-IoT)*, pp. 241–246, IEEE, 2014.

[7] F. Yang, S. Wang, J. Li, Z. Liu, and Q. Sun, "An overview of internet of vehicles," *China communications*, vol. 11, no. 10, pp. 1–15, 2014.

[8] J. Contreras-Castillo, S. Zeadally, and J. A. Guerrero-Ibañez, "Internet of vehicles: architecture, protocols, and security," *IEEE internet of things Journal*, vol. 5, no. 5, pp. 3701–3709, 2017.

[9] T. L. Koreshoff, T. Robertson, and T. W. Leong, "Internet of things: a review of literature and products," in *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration*, pp. 335–344, 2013.

[10] D. Metcalf, S. T. Milliard, M. Gomez, and M. Schwartz, "Wearables and the internet of things for health: Wearable, interconnected devices promise more efficient and comprehensive health care," *IEEE Pulse*, vol. 7, no. 5, pp. 35–39, 2016.

[11] G. Lu and W. H. Zeng, "Cloud computing survey," in *Applied Mechanics and Materials*, vol. 530, pp. 650–661, Trans Tech Publ, 2014.

[12] D. Pop, "Machine learning and cloud computing: Survey of distributed and saas solutions," *arXiv preprint arXiv:1603.08767*, 2016.

[13] W. Y. C. Wang, A. Rashid, and H.-M. Chuang, "Toward the trend of cloud computing," *Journal of Electronic Commerce Research*, vol. 12, no. 4, p. 238, 2011.

[14] K. Saharan and A. Kumar, "Fog in comparison to cloud: A survey," *International Journal of Computer Applications*, vol. 122, no. 3, 2015.

[15] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, 2016.

[16] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," 2015.

[17] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "Mobile edge computing: Survey and research outlook," *arXiv preprint arXiv:1701.01090*, 2017.

[18] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13–16, 2012.

[19] I. Stojmenovic and S. Wen, "The fog computing paradigm: Scenarios and security issues," in *2014 federated conference on computer science and information systems*, pp. 1–8, IEEE, 2014.

[20] S. Wang, G.-H. Tu, R. Ganti, T. He, K. Leung, H. Tripp, K. Warr, and M. Zafer, "Mobile micro-cloud: Application classification, mapping, and deployment," in *Proc. Annual Fall Meeting of ITA (AMITA)*, 2013.

[21] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1002–1016, 2016.

[22] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, 2019.

[23] J. Mills, J. Hu, and G. Min, "Communication-efficient federated learning for wireless edge intelligence in IoT," *IEEE IoT Journal*, 2019.

[24] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[25] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," *arXiv preprint arXiv:1404.2188*, 2014.

[26] W. De Mulder, S. Bethard, and M.-F. Moens, "A survey on the application of recurrent neural networks to statistical language modeling," *Computer Speech & Language*, vol. 30, no. 1, pp. 61–98, 2015.

[27] L. Zhang, S. Wang, and B. Liu, "Deep learning for sentiment analysis: A survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 4, p. e1253, 2018.

[28] A. Kamilaris and F. X. Prenafeta-Boldú, "Deep learning in agriculture: A survey," *Computers and electronics in agriculture*, vol. 147, pp. 70–90, 2018.

[29] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for iot big data and streaming analytics: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2923–2960, 2018.

[30] J. Liu, Y. Pan, M. Li, Z. Chen, L. Tang, C. Lu, and J. Wang, "Applications of deep learning to mri images: A survey," *Big Data Mining and Analytics*, vol. 1, no. 1, pp. 1–18, 2018.

[31] ETSI, "Mobile edge computing - introductory technical white paper," Sept. 2014.

[32] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, 2017.

[33] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[34] S.-H. Wang, P. P.-W. Huang, C. H.-P. Wen, and L.-C. Wang, "EQVMP: Energy-efficient and qos-aware virtual machine placement for software defined datacenter networks," in *The International Conference on Information Networking 2014 (ICOIN2014)*, pp. 220–225, IEEE, 2014.

[35] I. Althamary, C.-W. Huang, P. Lin, S.-R. Yang, and C.-W. Cheng, "Popularity-based cache placement for fog networks," in *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pp. 800–804, IEEE, 2018.

[36] B. Gao, Z. Zhou, F. Liu, and F. Xu, "Winning at the starting line: Joint network selection and service placement for mobile edge computing," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 1459–1467, IEEE, 2019.

[37] R. Mahmud, S. N. Srirama, K. Ramamohanarao, and R. Buyya, "Quality of experience (qoe)-aware placement of applications in fog computing environments," *Journal of Parallel and Distributed Computing*, vol. 132, pp. 190–203, 2019.

[38] T. He, N. Bartolini, H. Khamfroush, I. Kim, L. Ma, and T. La Porta, "Service placement for detecting and localizing failures using end-to-end observations," in *2016 IEEE 36th ICDCS*, pp. 560–569, IEEE, 2016.

[39] M. Turner and H. Khamfroush, "Meeting users' QoS in a edge-to-cloud platform via optimally placing services and scheduling tasks," in *2020 IEEE ICNC*, pp. 368–372, IEEE, 2020.

[40] T. Ouyang, R. Li, X. Chen, Z. Zhou, and X. Tang, "Adaptive user-managed service placement for mobile edge computing: An online learning approach," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 1468–1476, IEEE, 2019.

[41] T. He, H. Khamfroush, S. Wang, T. La Porta, and S. Stein, "It's hard to share: joint service placement and request scheduling in edge clouds with sharable and non-sharable resources," in *IEEE 38th ICDCS*, 2018.

[42] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 36, 2018.

[43] F. A. Salaht, F. Desprez, and A. Lebre, "An overview of service placement problem in fog and edge computing," *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, pp. 1–35, 2020.

[44] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, "Towards QoS-aware fog service placement," in *2017 IEEE 1st international conference on Fog and Edge Computing (ICFEC)*, pp. 89–96, IEEE, 2017.

[45] A. Yousefpour, A. Patil, G. Ishigaki, I. Kim, X. Wang, H. C. Cankaya, Q. Zhang, W. Xie, and J. P. Jue, "FogPlan: a lightweight QoS-aware dynamic fog service provisioning framework," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5080–5096, 2019.

[46] V. Farhadi, F. Mehmeti, T. He, T. F. L. Porta, H. Khamfroush, S. Wang, K. S. Chan, and K. Poularakis, "Service placement and request scheduling for data-intensive applications in edge clouds," *IEEE/ACM Transactions on Networking*, pp. 1–14, 2021.

[47] D. Xu, T. Li, Y. Li, X. Su, S. Tarkoma, and P. Hui, "A survey on edge intelligence," *arXiv preprint arXiv:2003.12172*, 2020.

[48] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, 2020.

[49] F. Manessi, A. Rozza, S. Bianco, P. Napoletano, and R. Schettini, "Automated pruning for deep neural network compression," in *2018 24th International Conference on Pattern Recognition (ICPR)*, IEEE, 2018.

[50] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5687–5695, 2017.

[51] P. S. Chandakkar, Y. Li, P. L. K. Ding, and B. Li, "Strategies for re-training a pruned neural network in an edge computing paradigm," in *2017 IEEE International Conference on Edge Computing (EDGE)*, pp. 244–247, IEEE, 2017.

[52] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, 2019.

[53] X. Zhao, M. Hosseinzadeh, N. Hudson, H. Khamfroush, and D. E. Lucani, "Improving accuracy-latency trade-off of edge-cloud computation offloading for deep learning services," in *IEEE Globecom Workshop on Edge Learning over 5G Networks and Beyond*, 2020.

[54] M. Hosseinzadeh, A. Wachal, H. Khamfroush, and D. E. Lucani, "Optimal accuracy-time trade-off for deep learning services in edge computing systems," in *IEEE International Conference on Communications*, 2021.

[55] Z. Lin, S. Bi, and Y.-J. A. Zhang, "Optimizing AI service placement and resource allocation in mobile edge intelligence systems," *arXiv preprint arXiv:2011.05708*, 2021.

[56] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, "Videoedge: Processing camera streams using hierarchical clusters," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 115–131, IEEE, 2018.

[57] L. Caccetta and A. Kulanoot, "Computational aspects of hard knapsack problems," *Nonlinear Analysis*, vol. 47, no. 8, pp. 5547–5558, 2001.

[58] D. Pisinger, "Where are the hard knapsack problems?," *Computers & Operations Research*, vol. 32, no. 9, pp. 2271–2284, 2005.

[59] S. Martello, D. Pisinger, and P. Toth, "New trends in exact algorithms for the 0–1 knapsack problem," *European Journal of Operational Research*, vol. 123, no. 2, pp. 325–332, 2000.

[60] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical society*, vol. 7, no. 1, pp. 48–50, 1956.

[61] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions—I," *Mathematical programming*, vol. 14, no. 1, pp. 265–294, 1978.

[62] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák, "Maximizing a submodular set function subject to a matroid constraint," in *International Conference on Integer Programming and Combinatorial Optimization*, pp. 182–196, Springer, 2007.

[63] S. Mitchell, S. M. Consulting, and I. Dunning, "PuLP: A linear programming toolkit for Python," 2011.

[64] johnjforrest, S. Vigerske, H. G. Santos, T. Ralphs, L. Hafer, B. Kristjansson, jpfasano, Edwin-Straver, M. Lubin, rlougee, jpgoncal1, h-i gassmann, and M. Saltzman, "coin-or/cbc: Version 2.10.5," Mar. 2020.

[65] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[66] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[67] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *arXiv preprint arXiv:1404.5997*, 2014.

[68] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

[69] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[70] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[71] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[72] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[73] H. Xu, W. Yu, D. W. Griffith, and N. Golmie, "A survey on industrial internet of things: A cyber-physical systems perspective," *IEEE Access*, vol. 6, pp. 78238–78259, 2018.

[74] X. Xu, M. ran Han, S. M. Nagarajan, and P. Anandhan, "Industrial internet of things for smart manufacturing applications using hierarchical trustful resource assignment," *Comput. Commun.*, vol. 160, 2020.

[75] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2017.

[76] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, pp. 1738–1762, 2019.

[77] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, 2016.

[78] D. Xu, T. Li, Y. Li, X. Su, S. Tarkoma, T. Jiang, J. Crowcroft, and P. Hui, "Edge intelligence: Architectures, challenges, and applications," *arXiv: Networking and Internet Architecture*, 2020.

[79] N. Hudson, M. J. Hossain, M. Hosseinzadeh, H. Khamfroush, M. Rahnamay-Naeini, and N. Ghani, "A framework for edge intelligent smart distribution grids via federated learning," *IEEE ICCCN*, 2021.

[80] V. Farhadi, F. Mehmeti, T. He, T. La Porta, H. Khamfroush, S. Wang, and K. S. Chan, "Service placement and request scheduling for data-intensive applications in edge clouds," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 1279–1287, IEEE, 2019.

[81] N. Hudson, H. Khamfroush, and D. E. Lucani, "QoS-aware placement of deep learning services on the edge with multiple service implementations," in *2021 IEEE ICCCN*, pp. 1–8, 2021.

[82] S. Hu and G. Li, "Dynamic request scheduling optimization in mobile edge computing for iot applications," *IEEE Internet of Things Journal*, vol. 7, pp. 1426–1437, 2020.

[83] T. Ma, M. Hempel, D. Peng, and H. Sharif, "A survey of energy-efficient compression and communication techniques for multimedia in resource constrained systems," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 963–972, 2012.

[84] X. Zou, X. Xu, C. Qing, and X. Xing, "High speed deep networks based on discrete cosine transformation," in *IEEE ICIP*, 2014.

[85] D. Fu and G. Guimaraes, "Using compression to speed up image classification in artificial neural networks," 2016. Technical report.

[86] D. Xu, Q. Li, and H. Zhu, "Energy-saving computation offloading by joint data compression and resource allocation for mobile-edge computing," *IEEE Communications Letters*, vol. 23, pp. 704–707, 2019.

[87] A. J. Hussain, A. Al-Fayadh, and N. Radi, "Image compression techniques: A survey in lossless and lossy algorithms," *Neurocomputing*, vol. 300, pp. 44–69, 2018.

[88] P. Tseng, "Convergence of a block coordinate descent method for nondifferentiable minimization," *Journal of optimization theory and applications*, vol. 109, no. 3, pp. 475–494, 2001.

[89] W. Feng, H. Liu, Y. Yao, D. Cao, and M. Zhao, "Latency-aware offloading for mobile edge computing networks," *IEEE Communications Letters*, vol. 25, pp. 2673–2677, 2021.

[90] S. Huang, C. Yang, S. Yin, Z. Zhang, and Y. Chu, "Latency-aware task peer offloading on overloaded server in multi-access edge computing system interconnected by metro optical networks," *Journal of Lightwave Technology*, vol. 38, pp. 5949–5961, 2020.

[91] I. A. Elgendy, W.-Z. Zhang, Y. Zeng, H. He, Y.-C. Tian, and Y. Yang, "Efficient and secure multi-user multi-task computation offloading for mobile-edge computing in mobile iot networks," *IEEE Transactions on Network and Service Management*, vol. 17, pp. 2410–2422, 2020.

[92] R. Yuan, B. Yang, Y. Liu, and L. Huang, "Modified gompertz sigmoidal model removing fine-ending of grain-size distribution," *Open Geosciences*, vol. 11, no. 1, pp. 29–36, 2019.

[93] B. Gompertz, "XXIV. On the nature of the function expressive of the law of human mortality, and on a new mode of determining the value of life contingencies. in a letter to francis baily, esq. frs &c," *Philosophical transactions of the Royal Society of London*, no. 115, pp. 513–583, 1825.

[94] S. Burer and A. N. Letchford, "Non-convex mixed-integer nonlinear programming: A survey," *Surveys in Operations Research and Management Science*, vol. 17, no. 2, pp. 97–106, 2012.

[95] W. E. Hart, C. D. Laird, J.-P. Watson, D. L. Woodruff, G. A. Hackebeil, B. L. Nicholson, J. D. Siirola, *et al.*, *Pyomo-optimization modeling in python*, vol. 67. Springer, 2017.

[96] H. Hui, Y. Ding, Q. Shi, F. Li, Y. Song, and J. Yan, "5g network-based internet of things for demand response in smart grid: A survey on application potential," *Applied Energy*, vol. 257, p. 113972, 2020.

[97] A. Faruqui, R. Hledik, and J. Tsoukalis, "The power of dynamic pricing," *The Electricity Journal*, vol. 22, no. 3, pp. 42–56, 2009.

[98] M. W. Ahmad, M. Mourshed, D. Mundow, M. Sisinni, and Y. Rezgui, "Building energy metering and environmental monitoring–a state-of-the-art review and directions for future research," *Energy and Buildings*, vol. 120, pp. 85–102, 2016.

[99] A. Taïk and S. Cherkaoui, "Electrical load forecasting using edge computing and federated learning," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2020.

[100] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *preprint arXiv:1610.02527*, 2016.

[101] A. Adeniran, M. A. Hasnat, M. Hosseinzadeh, H. Khamfroush, and M. Rahnamay-Naeini, "Edge layer design and optimization for smart grids," in *2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, pp. 1–6, IEEE, 2020.

[102] A. von Meier, D. Culler, A. McEachern, and R. Arghandeh, "Micro-synchrophasors for distribution systems," in *ISGT 2014*, pp. 1–5, 2014.

[103] A. von Meier, E. Stewart, A. McEachern, M. Andersen, and L. Mehrmanesh, "Precision micro-synchrophasors for distribution systems: A summary of applications," *IEEE Trans. on Smart Grid*, vol. 8, 11 2017.

[104] A. Shahsavari, M. Farajollahi, E. M. Stewart, E. Cortez, and H. Mohsenian-Rad, "Situational awareness in distribution grid using micro-pmu data: A machine learning approach," *IEEE Trans. on Smart Grid*, vol. 10, no. 6, pp. 6167–6177, 2019.

[105] S. Teimourzadeh, F. Aminifar, and M. Shahidehpour, "Contingency-constrained optimal placement of micro-pmus and smart meters in microgrids," *IEEE Trans. on Smart Grid*, vol. 10, no. 2, pp. 1889–1897, 2019.

[106] R. Siqueira de Carvalho, P. Kumar Sen, Y. Nag Velaga, L. Feksa Ramos, and L. Neves Canha, "Communication system design for an advanced metering infrastructure," vol. 18, p. 3734, Multidisciplinary Digital Publishing Institute, 2018.

[107] C. Pirak, T. Sangsuwan, and S. Buayairaksa, "Recent advances in communication technologies for smart grid application: A review," in *2014 IEECON*, pp. 1–4, IEEE, 2014.

[108] D. S. Kim, B. J. Chung, and Y. M. Chung, "Analysis of ami communication methods in various field environments," *Energies*, vol. 13, 2020.

[109] S. Finster and I. Baumgart, "Privacy-aware smart metering: A survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, 2014.

[110] N. Islam, I. Sultana, and M. S. Rahman, "Hkms-ami: A hybrid key management scheme for ami secure communication," in *Proceedings of International Conference on Trends in Computational and Cognitive Engineering* (M. S. Kaiser, A. Bandyopadhyay, M. Mahmud, and K. Ray, eds.), (Singapore), pp. 383–392, Springer Singapore, 2021.

[111] F. M. Cleveland, "Cyber security issues for advanced metering infrastructure (AMI)," in *2008 IEEE Power and Energy Society General Meeting-Conversion and Delivery of Electrical Energy in the 21st Century*, pp. 1–5, IEEE, 2008.

[112] C. Zhang, F. Luo, M. Sun, and G. Ranzi, "Modeling and defending advanced metering infrastructure subjected to distributed denial-of-service attacks," *IEEE Transactions on Network Science and Engineering*, 2020.

[113] M. A. Hasnat, M. J. Hossain, A. Adeniran, M. Rahnamay-Naeini, and H. Khamfroush, "Situational awareness using edge-computing enabled internet of things for smart grids," in *IEEE Globecom Workshops*, 2019.

[114] Y. Wang, Q. Chen, C. Kang, M. Zhang, K. Wang, and Y. Zhao, "Load profiling and its application to demand response: A review," *Tsinghua Science and Technology*, vol. 20, no. 2, pp. 117–129, 2015.

[115] D. Liang, L. Zeng, H.-D. Chiang, and S. Wang, "Power flow matching-based topology identification of medium-voltage distribution networks via ami measurements," *International Journal of Electrical Power & Energy Systems*, vol. 130, p. 106938, 2021.

[116] A. Amara Korba, N. Tamani, Y. Ghamri-Doudane, and N. E. I. karabadji, "Anomaly-based framework for detecting power overloading cyberattacks in smart grid ami," *Computers & Security*, vol. 96, p. 101896, 2020.

[117] L. Wei, L. P. Rondon, A. Moghadasi, and A. I. Sarwat, "Review of cyber-physical attacks and counter defense mechanisms for advanced metering infrastructure in smart grid," in *2018 IEEE/PES T&D*, 2018.

[118] M. J. Hossain and M. Rahnamay-Naeini, "Line failure detection from pmu data after a joint cyber-physical attack," in *IEEE PESGM*, 2019.

[119] T. Hu, Q. Guo, X. Shen, H. Sun, R. Wu, and H. Xi, "Utilizing unlabeled data to detect electricity fraud in AMI: A semisupervised deep learning approach," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3287–3299, 2019.

[120] S. Salinas, M. Li, and P. Li, "Privacy-preserving energy theft detection in smart grids: A P2P computing approach," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 9, pp. 257–267, 2013.

[121] M. Wen, D. Yao, B. Li, and R. Lu, "State estimation based energy theft detection scheme with privacy preservation in smart grid," in *2018 IEEE International Conference on Communications (ICC)*, pp. 1–6, 2018.

[122] S. Zhou, "The effect of smart meter penetration on dynamic electricity pricing: Evidence from the united states," *The Electricity Journal*, vol. 34, no. 3, p. 106919, 2021.

[123] A. Faustine, N. H. Mvungi, S. Kaijage, and M. Kisangiri, "A survey on non-intrusive load monitoring methodies and techniques for energy disaggregation problem," *ArXiv*, vol. abs/1703.00785, 2017.

[124] A. Zoha, A. Gluhak, M. Imran, and S. Rajasegarar, "Non-intrusive load monitoring approaches for disaggregated energy sensing: A survey.," *MDPI Sensors*, vol. 12(12), pp. 16838–16866, 2012.

[125] J. Z. Kolter and T. Jaakkola, "Approximate inference in additive factorial hmms with application to energy disaggregation," in *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics* (N. D. Lawrence and M. Girolami, eds.), vol. 22 of *Proceedings of Machine Learning Research*, (La Palma, Canary Islands), pp. 1472–1482, PMLR, 21–23 Apr 2012.

[126] B. Zhao, L. Stankovic, and V. Stankovic, "On a training-less solution for non-intrusive appliance load monitoring using graph signal processing," *IEEE Access*, vol. 4, pp. 1784–1799, 2016.

[127] J. Kelly and W. Knottenbelt, "Neural nilm: Deep neural networks applied to energy disaggregation," in *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, pp. 55–64, 2015.

[128] P. P. M. do Nascimento, "Applications of deep learning techniques on nilm," *Diss. Universidade Federal do Rio de Janeiro*, 2016.

[129] T.-T.-H. L. Jihyun Kim and H. Kim, "Nonintrusive load monitoring based on advanced deep learning and novel signature," *Computational Intelligence and Neuroscience*, vol. 2017, p. 22, 2017.

[130] V. Piccialli and A. M. Sudoso, "Improving non-intrusive load disaggregation through an attention-based deep neural network," *Energies*, vol. 14, no. 4, p. 847, 2021.

[131] H. Cao, S. Liu, R. Zhao, and X. Xiong, "IFed: A novel federated learning framework for local differential privacy in power internet of things," *International Journal of Distributed Sensor Networks*, vol. 16, no. 5, p. 1550147720919698, 2020.

[132] N. Felemban, F. Mehmeti, H. Khamfroush, Z. Lu, S. Rallapalli, K. S. Chan, and T. La Porta, "PicSys: Energy-efficient fast image search on distributed mobile networks," *IEEE Trans. on Mobile Computing*, 2019.

[133] V. Farhadi, F. Mehmeti, T. He, T. F. La Porta, H. Khamfroush, S. Wang, K. S. Chan, and K. Poularakis, "Service placement and request scheduling for data-intensive applications in edge clouds," *IEEE/ACM Transactions on Networking*, 2021.

[134] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, 2020.

[135] A. Singh, P. Vepakomma, O. Gupta, and R. Raskar, "Detailed comparison of communication efficiency of split learning and federated learning," *preprint arXiv:1909.09145*, 2019.

[136] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*.

[137] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *arXiv preprint arXiv:1812.06127*, 2018.

[138] A. Lalitha, O. C. Kilinc, T. Javidi, and F. Koushanfar, "Peer-to-peer federated learning on graphs," *preprint arXiv:1901.11173*, 2019.

[139] S. Savazzi, M. Nicoli, and V. Rampa, "Federated learning with cooperating devices: A consensus approach for massive IoT networks," *IEEE IoT Journal*, 2020.

[140] Z. Xu, Z. Yang, J. Xiong, J. Yang, and X. Chen, "Elfish: Resource-aware federated learning on heterogeneous edge devices," *preprint arXiv:1912.01684*, 2019.

[141] M. Ravanelli, P. Brakel, M. Omologo, and Y. Bengio, "Light gated recurrent units for speech recognition," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 2, pp. 92–102, 2018.

[142] V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, "A survey on security and privacy of federated learning," *Future Generation Computer Systems*, vol. 115, 2021.

[143] O. Parson, G. Fisher, A. Hersey, N. Batra, J. Kelly, A. Singh, W. Knottenbelt, and A. Rogers, "Dataport and NILMTK: A building data set designed for non-intrusive load monitoring," in *2015 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pp. 210–214, 2015.

[144] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[145] INRIX, "Congestion costs each American nearly 100 hours, $1,400 a year," Mar 2020. [Press release]. Retrieved from `https://inrix.com/press-releases/2019-traffic-scorecard-us/`.

[146] M. S. Anwer and C. Guy, "A survey of vanet technologies," *Journal of Emerging Trends in Computing and Information Sciences*, vol. 5, no. 9, pp. 661–671, 2014.

[147] S. K. Bhoi and P. M. Khilar, "Vehicular communication: a survey," *IET networks*, vol. 3, no. 3, pp. 204–217, 2014.

[148] Y. Chen, C. Li, W. Yue, H. Zhang, and G. Mao, "Engineering a large-scale traffic signal control: A multi-agent reinforcement learning approach," *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops*, pp. 1–6, 2021.

[149] M. A. Wiering, "Multi-agent reinforcement learning for traffic light control," in *Machine Learning: Proceedings of the Seventeenth International Conference (ICML'2000)*, pp. 1151–1158, 2000.

[150] L. Prashanth and S. Bhatnagar, "Reinforcement learning with function approximation for traffic signal control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 2, pp. 412–421, 2010.

[151] T. Chu, J. Wang, L. Codecà, and Z. Li, "Multi-agent deep reinforcement learning for large-scale traffic signal control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, pp. 1086–1095, 2020.

[152] M. Aslani, M. S. Mesgari, and M. Wiering, "Adaptive traffic signal control with actor-critic methods in a real-world traffic network with different traffic disruption events," *Transportation Research Part C-emerging Technologies*, vol. 85, 2017.

[153] X. Wang, L. Ke, Z. Qiao, and X. Chai, "Large-scale traffic signal control using a novel multiagent reinforcement learning," *IEEE Transactions on Cybernetics*, vol. 51, pp. 174–187, 2021.

[154] T. Tan, T. Chu, and J. Wang, "Multi-agent bootstrapped deep q-network for large-scale traffic signal control," *2020 IEEE Conference on Control Technology and Applications (CCTA)*, pp. 358–365, 2020.

[155] B. Abdulhai, R. Pringle, and G. J. Karakoulas, "Reinforcement learning for true adaptive traffic signal control," *Journal of Transportation Engineering-asce*, vol. 129, pp. 278–285, 2003.

[156] S. S. Mousavi, M. Schukat, and E. Howley, "Traffic light control using deep policy-gradient and value-function-based reinforcement learning," *IET Intelligent Transport Systems*, vol. 11, no. 7, pp. 417–423, 2017.

[157] P. G. Balaji, X. German, and D. Srinivasan, "Urban traffic signal control using reinforcement learning agents," *Iet Intelligent Transport Systems*, vol. 4, pp. 177–188, 2010.

[158] P. Zhou, X. Chen, Z. Liu, T. Braud, P. Hui, and J. Kangasharju, "DRLE: Decentralized reinforcement learning at the edge for traffic light control in the iov," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 4, pp. 2262–2273, 2020.

[159] S. El-Tantawy, B. Abdulhai, and H. Abdelgawad, "Multiagent reinforcement learning for integrated network of adaptive traffic signal controllers (MARLIN-ATSC): Methodology and large-scale application on downtown Toronto," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, pp. 1140–1150, 2013.

[160] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *preprint arXiv:1610.05492*, 2016.

[161] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, pp. 1273–1282, PMLR, 2017.

[162] R. Bellman, "A Markovian decision process," *Journal of mathematics and mechanics*, vol. 6, no. 5, pp. 679–684, 1957.

[163] P. Koonce and L. Rodegerdts, "Traffic signal timing manual.," tech. rep., United States. Federal Highway Administration, 2008.

[164] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *ArXiv*, vol. abs/1707.06347, 2017.

[165] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[166] H. H. Zhuo, W. Feng, Q. Xu, Q. Yang, and Y. Lin, "Federated reinforcement learning," 2019.

[167] B. Liu, L. Wang, and M. Liu, "Lifelong federated reinforcement learning: a learning architecture for navigation in cloud robotic systems," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4555–4562, 2019.

[168] N. Hudson, M. J. Hossain, M. Hosseinzadeh, H. Khamfroush, M. Rahnamay-Naeini, and N. Ghani, "A framework for edge intelligent smart distribution grids via federated learning," in *2021 International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–9, 2021.

[169] Y. J. Cho, J. Wang, and G. Joshi, "Client selection in federated learning: Convergence analysis and power-of-choice selection strategies," *arXiv preprint arXiv:2010.01243*, 2020.

[170] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of sumo-simulation of urban mobility," *International journal on advances in systems and measurements*, vol. 5, no. 3&4, 2012.

[171] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, "RLlib: Abstractions for distributed reinforcement learning," in *International Conference on Machine Learning*, PMLR, 2018.

[172] Z. Liu, "A survey of intelligence methods in urban traffic signal control," *IJCSNS International Journal of Computer Science and Network Security*, vol. 7, no. 7, 2007.

[173] M. Dotoli, M. P. Fanti, and C. Meloni, "A signal timing plan formulation for urban traffic control," *Control engineering practice*, vol. 14, no. 11, pp. 1297–1311, 2006.

[174] H. Ceylan and M. G. Bell, "Traffic signal timing optimisation based on genetic algorithm approach, including drivers' routing," *Transportation Research Part B: Methodological*, vol. 38, no. 4, pp. 329–342, 2004.

Vita

Nathaniel Hudson

Place of Birth

- Tacoma, Washington, United States of America

Education:

- University of Kentucky, Lexington, KY
  M.S. in Computer Science, May 2021
- Northern Kentucky University, Highland Heights, KY
  B.S. in Computer Science, May 2017
  Cum Laude

Professional Positions

- *Postdoctoral Scholar*, University of Chicago starting in August 2022
- *Graduate Research Assistant*, University of Kentucky Fall 2019–Spring 2022
- *Graduate Teaching Assistant*, University of Kentucky Fall 2017–Spring 2019
- *Academic Tutor*, Learning Assistance Programs, Northern Kentucky University Fall 2014–Spring 2017
- *Undergraduate Research Assistant*, University of North Texas May 2016–July 2016

Honors

- *2022 Diverse: Issues In Higher Education Rising Graduate Scholar*, one of the 10 selected for this honor.

- *Outstanding Student Paper Award ("Behavioral information diffusion for opinion maximization in online social networks")*, Department of Computer Science, University of Kentucky, 2021
- *Educational Diversity Scholarship*, Northern Kentucky University, 2013–2017
- *Project SOAR Scholarship*, Northern Kentucky University, 2013–2017
- *CINSAM Book Scholarship*, Northern Kentucky University, 2013–2017

Publications & Preprints

Peer-Reviewed Journal Publications

[1] **Nathaniel Hudson**, Hana Khamfroush, "Behavioral Information Diffusion for Opinion Maximization in Online Social Networks", IEEE Transactions on Network Science and Engineering, 2020.

[2] Hana Khamfroush, **Nathaniel Hudson**, Samuel Iloo, Mahshid Rahnamay-Naeini, "Influence Spread in Two-Layer Interdependent Networks: Designed Single-Layer or Random Two-Layer Initial Spreaders", Applied Network Science, 2019.

Peer-Reviewed Conference Publications

[3] **Nathaniel Hudson**, Pratham Oza, Hana Khamfroush, Thidapat Chantem, "Smart Edge-Enabled Traffic Light Control: Improving Reward-Communication Trade-off with Federated Reinforcement Learning", IEEE SMARTCOMP, 2022.

[4] Minoo Hosseinzadeh*, **Nathaniel Hudson***, Xiaobo Zhao, Hana Khamfroush, Daniel E. Lucani, "Joint Compression and Offloading Decisions for Deep Learning Services in 3-Tier Edge Systems", 2021 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN), 2021 (Equal contribution as first author)*.

[5] **Nathaniel Hudson**, Jakir Hossain, Minoo Hosseinzadeh, Hana Khamfroush, Mahshid Rahnamay-Naeini, Nasir Ghani, "A Framework for Edge Intelligent Smart Distribution Grids via Federated Learning", IEEE ICCCN, 2021.

[6] **Nathaniel Hudson**, Hana Khamfroush, Brent Harrison, Adam Craig, "Smart Advertisement for Maximal Clicks in Online Social Networks Without User Data", IEEE SMARTCOMP, 2020.

[7] Emory Hufbauer, **Nathaniel Hudson**, Hana Khamfroush, “A Proximity-Based Generative Model for Online Social Network topologies”, IEEE ICNC, 2020.

[8] **Nathaniel Hudson**, Matthew Turner, Asare Nkansah, Hana Khamfroush, “On the Effectiveness of Standard Centrality Metrics for Interdependent Networks”, IEEE ICNC, 2019.

Peer-Reviewed Workshop Publications

[9] Minoo Hosseinzadeh, **Nathaniel Hudson**, Sam Heshmati, Hana Khamfroush, “Communication-Loss Trade-Off in Federated Learning: A Distributed Client Selection Algorithm”, IEEE CCNC Secure FunctiON ChAining and FederaTed AI (SONATAI) Workshop, 2022.

[10] **Nathaniel Hudson**, Hana Khamfroush, Daniel E. Lucani, “QoS-Aware Placement of Deep Learning Services on the Edge with Multiple Service Implementations”, IEEE ICCCN Big Data and Machine Learning for Networking (BDMLN) Workshop, 2021.

[11] Xiaobo Zhao, Minoo Hosseinzadeh, **Nathaniel Hudson**, Hana Khamfroush, Daniel E. Lucani, “Improving Accuracy-Latency Trade-off of Edge-Cloud Computation Offloading for Deep Learning Services”, IEEE GLOBECOM Workshop on Edge Learning over 5G Networks and Beyond, 2020.