UNIVERSITY OF CALIFORNIA SAN DIEGO

Machine Learning Model Splitting on Mobile Edge Networks

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering (Communication Theory and System)

by

Song Wang

Committee in charge:

Professor Xinyu Zhang, Chair
Professor Sujit Dey
Professor Haojian Jin
Professor Patrick Pannuto
Professor Bhaskar Rao
Professor Ramesh Rao

2023

The dissertation of Song Wang is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2023

DEDICATION

To those who silently supported, quietly encouraged, and tirelessly believed in this pursuit.

EPIGRAPH

The purpose of computing is insight,

not numbers.

—Richard Hamming

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude and appreciation to all those who have contributed to the successful completion of this PhD thesis. Their unwavering support, guidance, and encouragement have been instrumental in shaping this journey and making this milestone possible.

I extend my heartfelt thanks to my esteemed supervisor, Professor Xinyu Zhang, for his invaluable mentorship throughout this research endeavor. His profound knowledge, constructive feedback, and continuous encouragement have been pivotal in honing my research skills and pushing me to strive for excellence.

I am immensely grateful to the members of my defense committee, Sujit Dey, Ramesh Rao, Bhaskar Rao, and Pat Pannuto. Their insightful feedback, critical evaluation, and constructive suggestions have immensely enriched the quality of this thesis. Their collective expertise and dedication to academic excellence have been a source of inspiration and motivation for me.

I would like to acknowledge the invaluable support and guidance provided by my internship mentors, Pengyu Zhang from Alibaba, Manikanta Kotaru, Xenofon Foukas, and Bozidar Radunovic from Microsoft Research. Their practical insights, industry knowledge, and encouragement during my internship significantly contributed to my research and added real-world relevance to my work.

I extend my gratitude to all my collaborators who shared their expertise and ideas during the course of this research. Their willingness to collaborate and engage in mean-

ingful discussions has broadened my understanding and enriched the outcomes of this study.

My deepest appreciation goes to my family for their unwavering love, encouragement, and understanding throughout this arduous journey. I want to express a special appreciation to my dear wife Jingqi Huang. Her unwavering support, patience, and belief in me have been the cornerstone of my resilience during challenging times. Her selflessness and sacrifices made it possible for me to focus on my research, and I am forever indebted to her.

I am grateful to my friends for their constant support, encouragement, and camaraderie. Their friendship and moments of levity provided a much-needed balance to the intensity of academic life and served as a reminder of the joys beyond research.

In conclusion, I want to acknowledge the collective efforts of all those mentioned above, as well as those who have supported me in ways beyond words. Your belief in my abilities and dedication to my success have been the driving force behind the completion of this PhD thesis. I am humbled and thankful for the opportunity to have undertaken this academic journey, and I hope to continue contributing to the field of research with the same passion and dedication that you have instilled in me. Thank you all.

Chapter 2 contains material from "HiveMind: Towards Cellular Native Machine Learning Model Splitting" by Song Wang, Xinyu Zhang, Hiromasa Uchiyama, and Hiroki Matsuda, which appears in the IEEE Journal on Selected Areas in Communications, 40(2):626–640, 2021. The dissertation author was the primary investigator and author of this paper.

Chapter 3 contains material from "NeuroMessenger: Towards Error Tolerant Distributed Machine Learning Over Edge Networks" by Song Wang and Xinyu Zhang, which appears in the IEEE International Conference on Computer Communications, 2022. The dissertation author was the primary investigator and author of this paper.

Chapter 4 contains material from "X-array: Approximating Omnidirectional Millimeter-Wave Coverage Using an Array of Phased Arrays" by Song Wang, Jingqi Huang, Xinyu Zhang, Hyoil Kim, and Sujit Dey, which appears in the ACM International Conference On Mobile Computing And Networking, 2020. The dissertation author was the primary investigator and author of this paper.

VITA

2018	B. E. in Electrical Engineering, Beijing University of Posts and Telecommunications
2018-2020	Research Assistant, University of California San Diego
2020	M. S. in Electrical Engineering, University of California San Diego
2020-2023	Research Assistant, University of California San Diego
2023	Ph. D. in Electrical Engineering, University of California San Diego

PUBLICATIONS

Song Wang and Xinyu Zhang, "NeuroMessenger: Towards Error Tolerant Distributed Machine Learning Over Edge Networks", *In IEEE INFOCOM*, 2022.

Song Wang, Xinyu Zhang, Hiromasa Uchiyama, and Hiroki Matsuda, "HiveMind: Towards Cellular Native Machine Learning Model Splitting", *IEEE Journal on Selected Areas in Communications*, 40(2):626–640, 2021.

Song Wang, Jingqi Huang, and Xinyu Zhang, "Demystifying Millimeter-Wave V2X: Towards Robust and Efficient Directional Connectivity Under High Mobility", *In ACM MobiCom*, 2020.

Song Wang, Jingqi Huang, Xinyu Zhang, Hyoil Kim, and Sujit Dey, "X-array: Approximating Omnidirectional Millimeter-Wave Coverage Using an Array of Phased Arrays", *In ACM MobiCom*, 2020.

ABSTRACT OF THE DISSERTATION

Machine Learning Model Splitting on Mobile Edge Networks

by

Song Wang

Electrical Engineering (Communication Theory and System)

University of California San Diego, 2023

Professor Xinyu Zhang, Chair

The rapid growth of Machine Learning (ML) model sizes poses challenges for mobile applications, especially when compared to the slower pace of mobile hardware development. Although cloud-based ML lightens this load by moving computations to Data Center (DC) servers, it struggles with limited bandwidth. On the other hand, Mobile Edge Computing (MEC)-based ML offers faster response times but can't always handle intense computations. To find a balance, *split ML* is introduced, which distributes ML tasks across various computing platforms.

The study delves into the inherent challenges of split ML, proposing innovative solutions: (1) HiveMind, A split ML system optimized for cellular networks; (2) NeuroMessenger, A mechanism that uses ML's inherent error tolerance to reduce data transmission delays; and (3) X-Array, An innovative radio architecture that meets split ML's high bandwidth needs. Together, these contributions seek to enhance the efficiency and feasibility of ML in the mobile computing landscape.

Chapter 1

Introduction

Machine learning (ML) models have become an increasingly integral part of mobile applications. Yet, as they continue to evolve, the sizes of these models have expanded exponentially. The Transformer model of 2017 required 65 million parameters [1], and the GPT-2.5 model of 2019 expanded to encompass 1.5 billion parameters. The most recent iteration, GPT-4, is projected to hold over 1.7 trillion parameters. Other models, for tasks such as image generation [2, 3, 4, 5] and video segmentation [6, 7], have also demonstrated similar exponential growth in size, as illustrated in Fig. 1.1. Meanwhile, the average memory size and computational capacity of mobile hardware have only grown linearly [8]. This discrepancy results in increased memory usage and power consumption, thereby inhibiting the operation of these models on mobile devices.

To address these constraints and enable the implementation of larger models on mobile platforms, developers have started employing cloud-based ML schemes [9, 10, 11]. As shown in Fig. 1.2, in cloud-based ML, the inference of the ML model is hosted on Data

Center (DC) servers [9, 10]. Applications upload their input data to the DC server and retrieve the corresponding inference results. By offloading the computational and storage demands of ML models to the DCs, this system bypasses the hardware constraints of mobile devices, enabling a better Quality-of-Service (QoS) through leveraging the superior computational power of DC servers [12, 13].

Nonetheless, cloud-based ML faces scalability challenges. The networking capacity of DCs is insufficient to support the transmission of mobile users' input data to ML servers. As evidenced by a 2021 CISCO study, end devices generated nearly 850ZB of data annually, while global DC traffic could only handle 20.6ZB [14]. This bandwidth shortfall increases communication latency and undermines QoS.

To address the limitations of cloud-based ML, the concept of Mobile Edge Computing (MEC)-based ML has been proposed, capitalizing on the rapidly expanding MEC infrastructures which deploy data centers close to users, offering cloud-like services [12, 13, 11, 15]. Compared to traditional cloud data centers (DC), MEC deployment offers enhanced flexibility. With its compact nature, MEC is not bound by rigid infrastructure requirements and can even be situated adjacent to base stations. Consequently, it augments computational capacity beyond what the standard DC provides. In addition, due to their closer proximity to users, MEC servers reduce the communication latency for ML input data significantly compared to cloud-based ML [11, 15]. However, MEC servers are generally less powerful than DC servers, and as such, the computational latency of MEC-based ML has yet to reach the levels achieved by cloud-based ML.

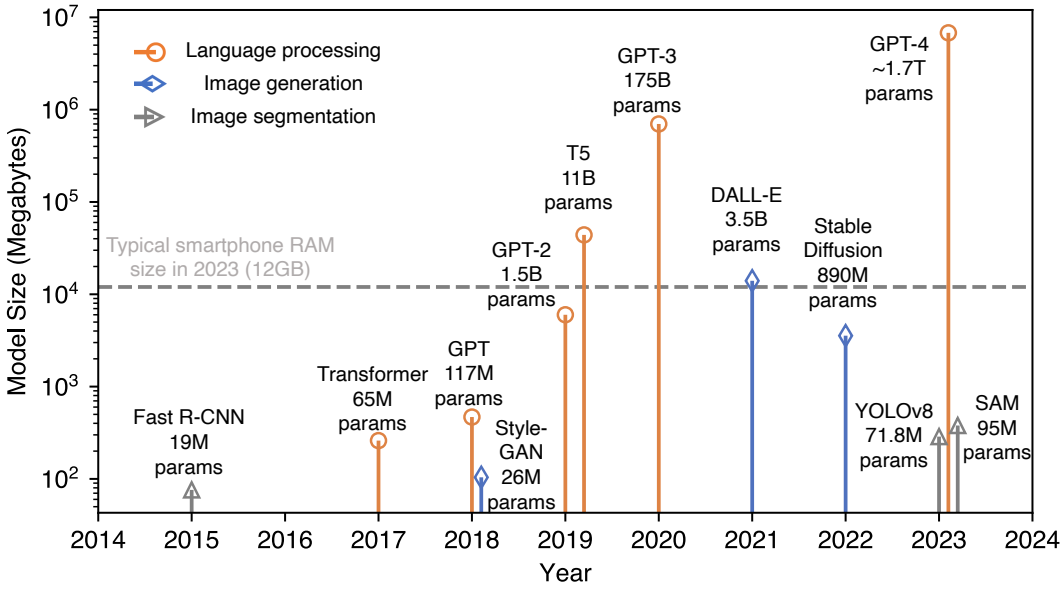


Figure 1.1: The exponential growth trend of ML model sizes, in comparison to the mobile hardware limitations.

1.1 Split ML: Opportunities and Challenges

The simultaneous utilization of the communication advantages of Mobile Edge Computing (MEC)-based Machine Learning (ML) and the computational strengths of cloud-based ML necessitates a novel approach, namely split ML [11, 16]. The concept of split ML, depicted in Fig. 1.2, mirrors the idea of pipeline parallelism inherent in traditional distributed ML systems. A split ML system dissects a mobile ML model inference into multiple segments, allocating them to various computing units including the mobile device, MEC servers, and cloud servers. Split ML can offload ML computations to MEC servers during poor network conditions and to cloud servers when the network is favorable. Consequently, it circumvents the latency bottleneck that both MEC-based and cloud-based

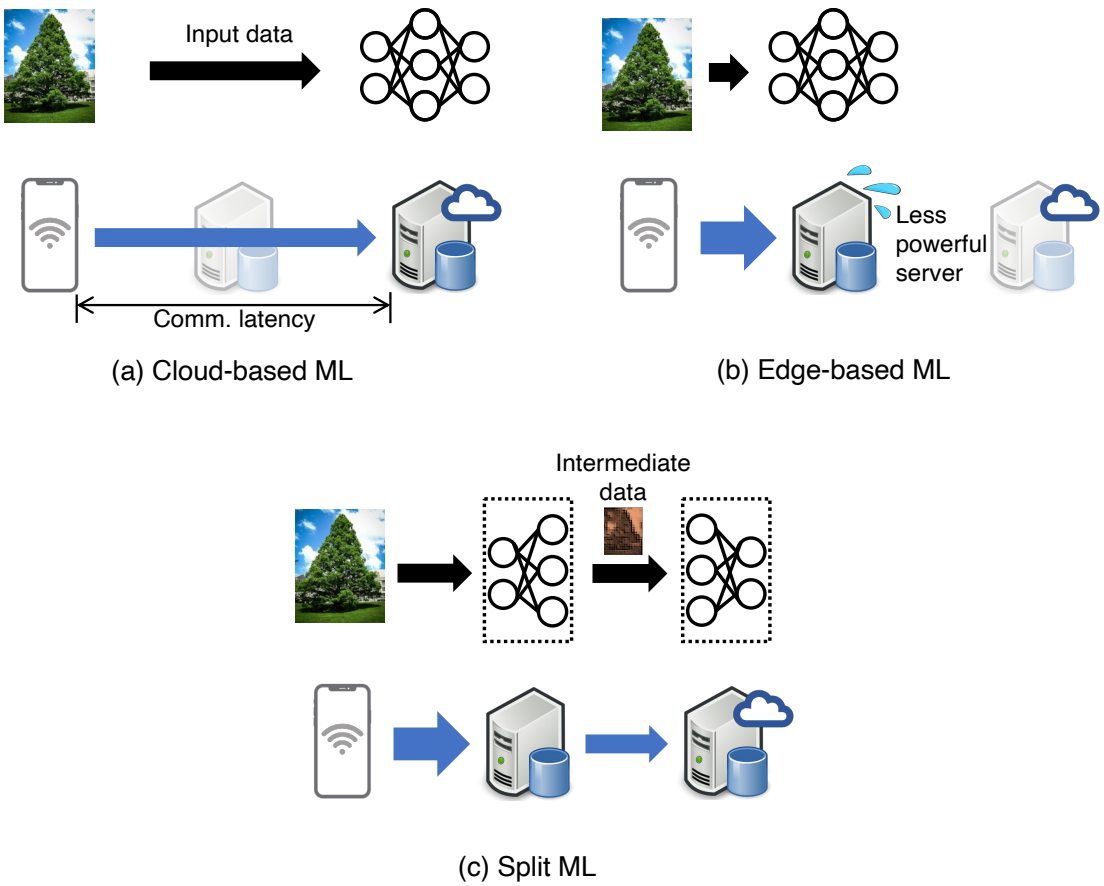


Figure 1.2: An architectural comparison of (a)cloud-based ML, (b)MEC-based ML, and (c)Split ML

ML might cause.

Designing a split ML system, however, is far from trivial. The distributed and pipelined model within split ML presents new challenges to both the splitting mechanism and the underlying communication hardware. This dissertation identifies three key challenges associated with split ML:

(i) *Allocation of ML model components:* A typical neural network comprises hundreds, sometimes thousands of layers. Splitting it across several MEC/cloud servers could

present $10^6 - 10^9$ potential options. It is clearly not feasible to brute-forcily assess all split options since split ML system needs to make rapid decisions to accommodate network dynamics. Hence, a light-weight algorithm to obtain the optimal split option in real-time is necessary. Moreover, the evaluation of split options often involves multiple metrics, such as end-to-end latency and energy consumption, requiring the split algorithm to accommodate these metrics simultaneously. Lastly, certain ML models lacking linear structures, for instance, multi-agent reinforcement learning models and federated learning models, require specific splitting algorithms as they cannot be divided layer-by-layer.

(ii) *Efficient transmission of ML intermediate data:* Split ML involves the exchange of intermediate output from the middle layers of an ML model between servers. This intermediate data, which can range from several megabytes to hundreds of megabytes [16], presents significant transmission challenges on traditional wireless communication stacks. For example, the transmission of large data under dynamic wireless link conditions is prune to frequent retransmissions, leading to increased communication latency. Previous research has demonstrated the unique error tolerance capabilities of ML intermediate data in distributed ML training [17]. As such, the underlying communication stack of split ML applications should adapt to these error tolerance capabilities to optimize efficiency.

(iii) *Design of communication hardware for split ML bandwidth requirements:* Split ML demands a throughput at the gigabit-per-second (Gbps) level with millisecond-level latency [11]. Such requirements can only be fulfilled by millimeter-wave (mmWave) radios. However, mmWave radios are plagued by limited coverage due to severe attenuation, restricting their range to around 200 meters—a distance far smaller than the typical gap

between two MEC sites [18]. Moreover, the Field-of-View (FoV) of an mmWave radio only covers approximately 120◦, thereby limiting user mobility. Thus, an omni-coverage mmWave radio is necessary for split ML with general mobile users.

1.2 Dissertation Contributions

In this dissertation, we thoroughly examine the system designs of split ML. We identify the fundamental challenges associated with implementing a practical split ML system and utilize a range of techniques — encompassing a novel split algorithm, innovative communication software, and hardware designs—to realize an effective split ML system on mobile edge networks. The primary contributions of this dissertation are:

(i) In Chapter 2, we introduce HiveMind, the first practical multi-split ML system tailored for 5G cellular networks. HiveMind reformulates the complicated multi-split problem to a min-cost graph search and optimizes the distributed algorithm to drastically reduce the signaling overhead. Benefit from its low overhead property, HiveMind makes the optimal split decision on multiple computing nodes in real-time and adapts the split decisions to the instantaneous network dynamics. HiveMind also incorporates a multi-objective mechanism that accommodates heterogeneous objectives for a single ML task. HiveMind adapts to a wide range of ML frameworks, including non-linear models like Recurrent Neural Network (RNN), Federated Learning (FL), and Multi-agent Reinforcement Learning (MARL). We evaluate HiveMind on 5G MEC network simulators with realistic traffic patterns and real-life MEC computation/communication profiles. Our experiments

demonstrate that HiveMind achieves the optimal efficiency comparing to state-of-art split ML designs.

(ii) In Chapter 3, we first characterize the error tolerance capability of state-of-art distributed ML frameworks. Based on the observations, we propose NeuroMessenger, a lightweight mechanism that can be built into the cellular network stack, which can enhance and utilize the error tolerance in ML data to reduce communication overhead. NeuroMessenger does not require per-model profiling and is transparent to application layer, which simplifies the development and deployment. Our experiments on a 5G simulation framework demonstrate that NeuroMessenger reduces the end-to-end latency by up to 99% while maintaining less than low accuracy loss under various link conditions.

(iii) In Chapter 4, we propose X-Array, a first-of-its-kind omni-directional mmWave radio architecture to support the gigabyte bandwidth requirement of split ML. X-Array jointly selects the arrays and beams, and applies a dynamic co-phasing mechanism to ensure different arrays' signals enhance each other. X-Array also incorporates a link recovery mechanism to identify alternative arrays/beams that can efficiently recover the link from outage. We have implemented X-Array on a commodity 802.11ad APA radio. Our experiments demonstrate that X-Array can approach omni-directional coverage and maintain high performance in spite of link dynamics.

Chapter 2

Cellular Native Machine Learning Model Splitting

2.1 Introduction

The booming mobile Machine Learning (ML) applications are challenging the current computing and communication network architectures. Amid the rapid maturity of mobile machine learning platforms, *e.g.*, Google ML kit [19], Apple Core ML [20], and Fritz AI [21], more than 10% of mobile apps have incorporated ML models, with use cases ranging from face identification, object detection, to intelligent personal assistants and augmented reality [22]. Recent studies also proposed to integrate ML into 5G networks to optimize network functions such as QoS-aware routing, resource allocation, and slice management [23, 24, 25, 26]. In addition, the emerging 6G is envisioned to bring human-like intelligence into every aspect of networking systems [27]. However, due to the computa-

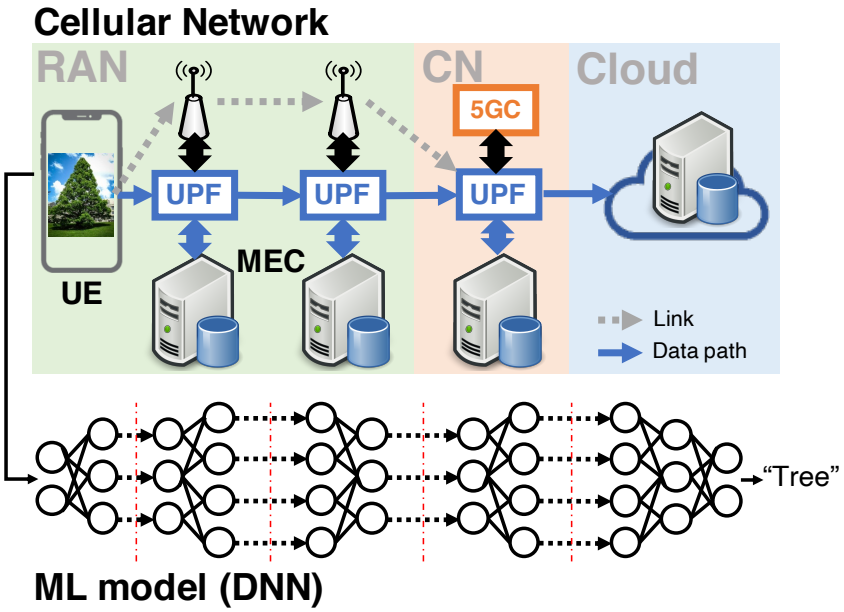


Figure 2.1: An example of 5G cellular native ML: An UE, three MEC nodes, and a cloud server form a 5-hop MEC chain. Each device executes a part of the ML model to a certain layer and send the intermediate data to the next device on the chain.

tion resource constraints on mobile devices, such mobile ML applications typically only use miniature models hundreds of times smaller than standard ML models [22], which hampers model accuracy and limits their use cases. The stringent computation power budget further renders the more computation-heavy ML training tasks impossible. On the other hand, offloading these tasks to the cloud may incur high data transfer overhead and sometimes can be even slower than on-device computing [16].

To enable computation-intensive mobile ML applications, recent work explored edge computing infrastructures [15] that offer cloud-like services within the cellular network. To minimize latency without compromising model accuracy, such edge ML implementations can split a model into multiple parts, and allocate them among different computing nodes,

including the user equipment (UE), mobile edge computing (MEC) servers, and cloud servers. For the commonly used deep neural network (DNN) model, for example, each part corresponds to multiple DNN layers. Each node executes the model up to a specific layer, and sends the intermediate data to the next node. With such UE-edge-cloud synergy, a split ML system can dynamically assign parts of a model to the computing nodes based on network conditions and computation resources, to alleviate the pressure of computation on UE devices and potentially optimize end-to-end latency and energy consumption [12, 13, 28, 16, 29].

Existing research abstracts the ML model splitting as redistributing the computing load across a generic client-server link. As the 5G network infrastructure evolves to embrace built-in computing capabilities, an important question arises: Can the 5G networking and computing stack itself natively support AI/ML through model splitting? Unlike in the abstract model, a single 5G site often consists of many MEC servers distributed across different vantage points in the RAN/core network. Splitting the ML model across such a unique distributed system, potentially involving dynamic links and UEs with different performance objectives, becomes a non-trivial problem.

More specifically, such a cellular-native model splitting needs to address three unique challenges. ***(i)*** *Multi-split ML models over a 5G network.* 5G's native MEC support [15] and flexible traffic steering capabilities can enable a new *multi-split* scheme among the UE, multiple MEC servers, and the cloud server. Fig. 2.1 shows a typical case of ML inference model splitting, where an image classification NN is split across a UE, three MEC nodes, and a cloud server. The intermediate output from each partition of

the NN is transmitted to the next computing node via wireless backhaul or wired links. The inference result is output at the last partition of the network. In such multi-split scenarios, the number of split options grows exponentially with the number of ML layers and computing nodes, and are often on the order of millions. In addition, the split decision needs to accommodate the computing resources distributed across the network, along with varying network conditions. Due to all such complexities, the linear searching method in existing ML model splitting designs cannot be applied to multi-split systems.

(ii) *Multi-objective split.* Compared with a single atomic model, the split ML decision over 5G networks should be optimized to flexibly accommodate different objectives, *e.g.*, inference/training latency, energy consumption, and privacy preservation. Some of these objectives are based on best efforts, *i.e.* maximization/minimization, whereas some are quality assurances, *i.e.*, ensuring a performance metric does not exceed a predefined threshold. Accommodating such heterogeneous objectives simultaneously poses a new challenge for the splitting decision making.

(iii) *Splitting for non-linear ML models.* Existing single-split approaches are limited to standard *linear* ML models with a chain of layers. Thus, a simple linear search across all inter-layer cuts suffices to identify the optimal split point. However, other commonly used ML paradigms, *e.g.*, Recurrent Neural Network (RNN) and Collaborative Learning, require additional information transfer between the same or different modules. Directly applying split ML on them fails to account for the extra communication overhead and may result in highly suboptimal performance.

In this paper, we propose *HiveMind*, a novel multi-split ML framework that ad-

dresses the aforementioned challenges through three design choices. *(i)* A distributed split ML algorithm. We first reformulate the multi-split problem into a min-cost graph search. To avoid the huge communication overhead that renders the existing solutions infeasible, we propose a distributed min-cost graph algorithm tailored for 5G MEC networks. Through graph pruning and information aggregation, our algorithm dramatically cuts down the number of inter-node signaling messages, thus enabling an efficient and practical multi-split without the loss of optimality on split decisions. *(ii)* A mechanism to simultaneously accommodate best efforts objectives (*e.g.*, minimizing energy cost) and quality assurance objectives (*e.g.*, latency threshold) in the splitting decision. The mechanism discriminates the quality assurance metrics with a non-linear mapping function, and enforces the quality assurance objectives without compromising the optimality of the best effort metrics . *(iii)* Splitting non-linear ML models. We further broaden the application domain of our split ML algorithm by extending it to non-linear ML models, including recurrent ML models and collaborative ML models, whereas the latter involves Multi-agent Reinforcement Learning (MARL) and Federated Learning (FL). Our solution takes into account the iterative feedback structures commonly seen in such models while requiring little modifications to the standard algorithm.

We evaluate HiveMind on a 5G network simulation framework, which represents a tree-structured integrated access and backhaul (IAB) network and edge/cloud computing devices co-located on all IAB gNBs, CN, and cloud server, in compliance with 3GPP's provisioning of 5G MEC architecture [30, 11]. The evaluation framework adopts synthetic traffic traces that faithfully reproduce the traffic characteristics of a cellular network. Our

experimental results demonstrate that *(i)* HiveMind is able to adapt to a wide range of traffic load. It outperforms cloud and UE-based baselines by up to 89.8% under high traffic load. HiveMind benefits more from MEC capability gains than the baselines by up to 47.2%, especially from the MECs co-located with IAB gNBs. *(ii)* HiveMind can simultaneously accommodate the best effort and quality assurance objectives, and outperforms the heuristic linear multi-objective by 22.9% on the best effort objective. *(iii)* HiveMind reduces the parameter feedback latency on both RNN and collaborative learning models, and outperforms the standard split by up to 2.3× on multiple criteria.

HiveMind, to our knowledge, marks the first practical multi-split ML system tailored for 5G MEC networks. Its contributions can be summarized as follow: (i) A novel cellular native split ML algorithm that enables the practical multi-split ML by distributively optimizing split assignment with negligible overhead. (ii) A multi-objective mechanism that adapts different types of objectives to a single multi-split task. (iii) Extension of the multi-split algorithm to widely adapted non-linear ML models including RNN and collaborative ML. (iv) Validation of HiveMind on a 5G simulator against state-of-art ML splitting designs.

2.2 Related Work

2.2.1 Distributed ML

Recent research explored distributed machine learning to reduce the processing time of mobile ML applications leveraging edge or cloud computing devices. Ho *et al.*

[31] proposed to use a centralized parameter server to aggregate the local gradient, and schedule training tasks on the local nodes. Agarwal *et al.* [32] designed AllReduce that further extends this paradigm to a tree structure, by accumulating and passing the local gradient from child nodes to parent nodes. Foster *et al.* [33] introduced a fully distributed paradigm where each node broadcasts the local gradient to all other nodes. In addition to the latency-oriented parallel computing paradigms, Konevcny *et al.* [34, 35, 36, 37, 38, 39] proposed Federated Learning (FL) framework with an emphasis on preserving data privacy. FL obscures the local update from local nodes so that the parameter server cannot infer sensitive information, but can still keep the training accuracy. The above distributed ML designs assume a client/server architecture, where each computing client trains one instance of the whole model. In contrast, our split ML framework partitions an ML model so that the different parts are executed sequentially on different computing nodes within a cellular network. In addition, distributed ML mainly focuses on ML training. Our split ML framework can be applied to both training and inference.

2.2.2 ML model splitting.

ML model splitting has garnered much interest in the past two years. In particular, the 3GPP standardization group recognized the performance benefits for ML model splitting and has been investigating protocol-level primitives to support ML model splitting within the cellular edge/core networks [11]. Much of the related research verified the advantages of ML model splitting and focused on partitioning the DNN computing load to meet certain optimization objectives. Kang *et al.* [16] proposed to identify a single

splitting point to cut a DNN inference model in two parts, executed by a generic client and server respectively, to optimize running latency or energy consumption. Hu *et al.* [40] further extended the single split scheme to DNN models with directed acyclic graph representation. The single-split approach splits between the UE and one server, and the performance improvement is largely limited by the computation resource and the link condition of the server machine. In contrast, our multi-split approach can flexibly assign the ML model to multiple MECs in order to adapt to dynamic link and computation resources. Narayanan *et al.* [41] proposed an optimization-driven split ML framework, PipeDream, that assigns parts of a model to multiple GPUs to reduce training latency on a single machine. Among the existing research in distributed/parallel ML, PipeDream shares the most similarity with our work. However, PipeDream assumes static links between GPUs where the partitioning is done once and for all. The linear programming based optimizer itself incurs around 8 s running latency on a server-level machine [41], which renders it unsuitable for real-time splitting of ML *inference* models in dynamic cellular networks.

2.2.3 Tailoring ML models to edge computing systems.

Besides model distribution and partitioning, existing work also explored other mechanisms to customize ML models for edge computing. Teerapittayanon *et al.* [42] proposed an early-exit mechanism, BranchyNet, which adds exit points in the middle of an ML model to cut the inference delay at the cost of lower accuracy. A follow-on distributed ML design, DDNN [43], further proposed that each end device sends the output of its exit point to an edge server which performs aggregation. Since the early exit mechanism

skips part of the model structure, the inference accuracy is largely compromised. So far no early exit design achieves more than 80% of inference accuracy from the early exit points comparing to the full model [42, 43, 44, 45, 46, 47]. Besides, it is feasible to compress the intermediate data transfer between UE and cloud in order to reduce communication latency [48, 49, 50, 51, 52, 53, 54, 55]. The state-of-art intermediate data compression design [55] achieves up to 5000/1 compression ratio by training a model-specific NN compressor while suffering a relative small accuracy loss of 8% due to the information loss during compression. Comparing to above two mechanisms, our split ML framework does not modify the model structure or intermediate data and reduces latency without sacrificing the inference accuracy. Note that the early-exit and compressing approaches are not in conflict with our split design. Instead, it is possible to apply them on top of HiveMind, *i.e.* adding early exit points at the split points or compressing the intermediate data out of the split points, to further optimize the processing latency.

2.3 The Need for Multi-Split in 5G MEC Networks

In this section, we explain the motivation for adopting multi-split ML and demonstrate its advantages with an example scenario. In a mobile ML application, the total overhead is attributed to two factors: the computation overhead in running the neural network layers, and the communication overhead in transmitting data between computing nodes, *e.g.*, UE uploading an input image to the cloud server. Conventional NN model runs on either UE or cloud server. UE-based ML models execution suffers from large compu-

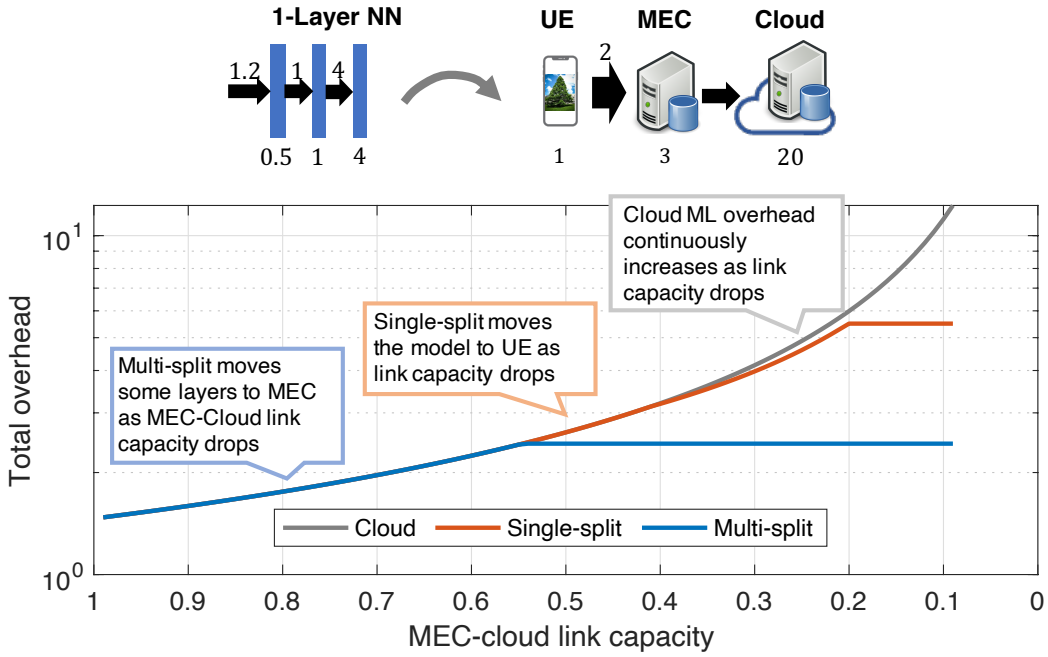


Figure 2.2: The latency comparison of 4 split architectures: (1) UE computing, (2) Cloud single split, (3) Edge single split, (4) Multi-split on UE, edge, and Cloud

tation overhead due to its stringent computation budget and cloud-based ML often incurs large communication overhead due to limited communication link capacity. In comparison, split ML achieves a flexible tradeoff between computation overhead and communication overhead by dynamically splitting a ML model between the UE and cloud [16, 40, 30]. It can achieve low computation overhead when the link capacity is high and avoid high communication overhead otherwise. In addition, comparing to existing single-split schemes [16, 40, 30], multi-split ML further improves the efficiency by leveraging a chain of MEC servers, striking a middle ground between UE and cloud with more computation budget than UE-only and a faster and more stable communication link than cloud-only model execution.

To showcase the advantage of multi-split ML, we consider a simple scenario where a UE, MEC, and cloud server split a 3-layer NN, as shown in Fig. 2.2. The layers have computation loads of 0.5, 1, and 4 units and communication load of 1.2, 1, and 4 units. The three computation nodes have computation capacity of 1, 3, and 10 units respectively and the link between UE and MEC has a capacity of 2 units. We assume a simple overhead model where the overhead is load divided by capacity, *e.g.*, transmitting input on UE-MEC link takes $\frac{1.2}{2} = 0.6$ unit time. The link capacity between MEC and cloud is set to vary between $\frac{1}{10}$ to 1 to simulate the link dynamics. We compare multi-split ML with two schemes: cloud ML, where UE always upload the input to the cloud for computation, and single-split ML, where the model is split between UE and the cloud. Fig. 2.2 shows the total overhead of three schemes as the MEC-cloud link capacity decreases from 1 to $\frac{1}{10}$. We see cloud ML's overhead continuously increases as the communication overhead

increases (*i.e.*, link capacity decreases). Single-split ML starts with all layers assigned to the cloud and transfers the layers to UE when link capacity further drops, to avoid increasing communication overhead. Mutli-split ML achieves a even lower total overhead on top of single-split ML by assigning the last two layer to MEC instead of UE. This experiment shows that multi-split avoids the increasingly large communication overhead in cloud ML and achieves a lower computation latency than single-split ML. Note that this experiment only demonstrates an simplified typical scenario. In Sec. 2.7, we will show that in a more detailed and realistic 5G setting, our multi-split ML reduces total latency by 37 to 90% compared with cloud ML and 32 to 55% compared with single-split ML.

2.4 HiveMind Multi-Split Design

In this section, we first provide a primer on the 5G MEC system that enables the multi-split ML framework. Then we introduce HiveMind multi-split design for both ML inference and training, and its runtime optimization under network dynamics.

2.4.1 A Primer on 5G MEC for ML

Similar to the legacy 4G architecture, 5G networks separate the Radio Access Network (RAN) and Core Network (CN) functions. Yet Integrated Access and Backhaul (IAB) is introduced in the RAN as a unique feature, where basestations (*i.e.*, gNBs) form a tree-like topology with multi-hop wireless backhaul links [30]. Due to the flexible deployment of data plane, ETSI group [15] provisions a flexible deployment of MECs at different

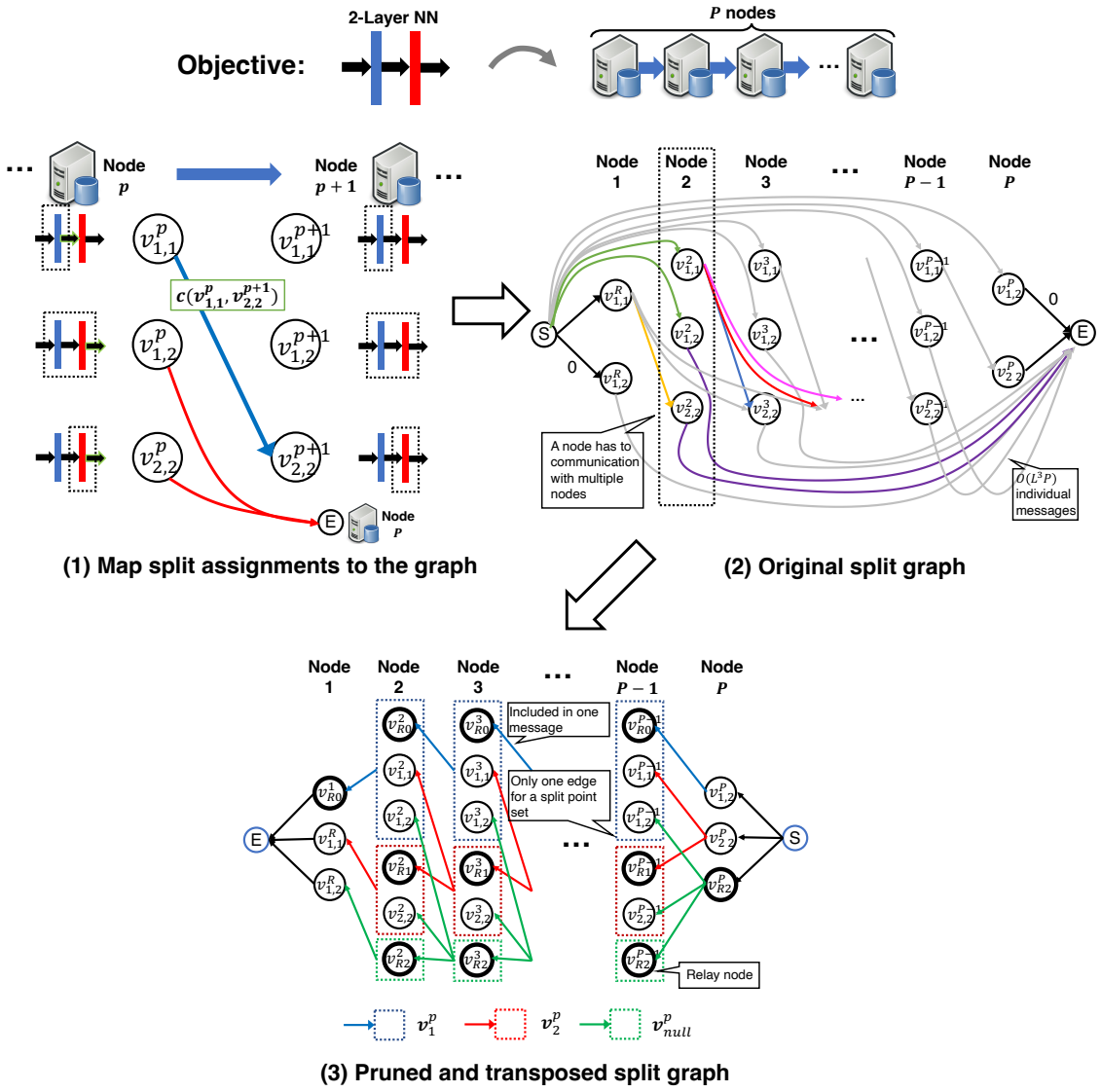


Figure 2.3: Graph representation of the HiveMind multi-split: (1) Map split assignments to graph. (2) The original split graph representation has numerous edges. (3) The pruned and transposed graph limits the edges strictly between adjacent MEC nodes.

vantage points within the 5G infrastructure, including the gNBs, RAN aggregation point, and the core network site. The flexible MEC deployment provides mobile applications with easier and faster access, especially for the gNB MECs which can be reached by UEs in one hop. The communication overhead is hence expected to be much shorter than remote cloud access [12], rendering it feasible to accelerate ML inference/training. On the other hand, the 5G User Plane Function (UPF) enables the free steering and routing of application traffic among UEs and MECs attached to different network entities [18], allowing the formation of multi-hop MEC chains. All above 5G features jointly enable the multi-split ML paradigm where an ML model is split into multiple parts and assigned to a chain of MEC nodes, as shown in Fig. 2.1.

2.4.2 Problem formulation

The multi-split problem. We now describe the multi-split problem formulation. For simplicity, we assume a linear DNN and a single objective of minimizing inference latency. In later sections, we will extend the design to non-linear ML models and multi-objectives.

Consider a scenario where $P-2$ MEC nodes in 5G system, along with a UE and a cloud server, form a P-node MEC chain where the first node $p = 1$ is the UE and the P-th node $p = P$ is the cloud server. The UE, serving as the *source node*, initiates a split ML task that utilizes the MECs along the route from UE to the cloud. We refer to the cloud server as the *sink node*, as it is the last node along the chain that may undertake part of the ML processing load. The ML model to be split across the network has L layers. We

define a split decision as two functions $u(p) = m, w(p) = n$ to represent assigning layers m to n to the node p. Suppose the layer-wise computation latency τ_l^p and communication latency ϵ_l^p of transferring intermediate data are known to all P nodes through profiling [16], the problem of finding the optimal split decision to minimize the total latency can be expressed as follow:

$$\min_{u,w} \quad \sum_{p=1}^{P} \sum_{l=u(p)}^{w(p)} \tau_l^p + \sum_{p=1}^{P} \epsilon_{w(p)}^p \tag{2.1}$$

$$\text{s.t.} \quad u(p) \leq w(p), \forall p \tag{2.2}$$

$$u(p) = w(p-1) + 1, 2 \leq p \leq P \tag{2.3}$$

$$w(P) = L \tag{2.4}$$

The first term in Eq. (2.1) sums up the computation latency of all P nodes and the second term sums up the communication latency of transferring intermediate data across adjacent nodes. Eq. (2.2)-(2.4) ensure the assignment includes all L layers in the ML model in correct order without overlapping. Note that it is valid to "skip" a node by assigning no layer to it. In such cases, the communication latency over the skipped node still needs to be included since within the IAB network the intermediate data has to travel through the skipped node.

Mapping split assignments to graph. The above optimization framework uses functions $u(p), w(p)$ as variables. Although they can be treated as vectors to fit into existing integer programming solutions, the mapping between them and the computation latency, *i.e.* $\tau_{u(p)}^{w(p)}$, is a non-convex function since τ_l^p is arbitrary. Hence, it is hard to solve the problem directly with integer programming. If we take the brute-force approach

and examine all possible split options, then it requires calculating the latency for all $\binom{P+L-1}{L}$ split options (split L layers into P sets while allowing 0 layer in a set since it is possible not to assign layers to a node), *e.g.*, a ResNet50[56] split on 5 nodes has 4.0×10^7 options. Examining such a huge amount of split options requires significant processing time, let alone the overhead caused by gathering the latency profiles and distributing the split decisions to all MECs. Hence a direct search, as done in existing single-split solutions, cannot meet the real-time requirement of cellular native split ML. To overcome these limitations, we modify the variables and reformulate the problem as a classic linear optimization - shortest path problem. We first convert the search space for the split decision into a directed graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$. The set of vertices:

$$\mathbf{V} = \{v^p_{m,n} | \forall p, 1 \leq m \leq n \leq L\} \tag{2.5}$$

embodies all possible assignment decisions on all P nodes, where a single vertex $v^p_{m,n}$ represents the decision of assigning layers m to n to node p. To avoid confusion, we use the term "vertex" to refer to the vertices in the graph and "node" to refer to the computing nodes in the cellular network. Then we can easily connect the vertex following the constraints in Eq. (2.2)-(2.4):

$$\begin{aligned} \mathbf{E} = \{(v^p_{m,n}, v^q_{x,y}) | p < q, m \leq n, x \leq y, \\ x = n+1, y = L \text{ when } q = P\} \end{aligned} \tag{2.6}$$

where an edge $(v^p_{m,n}, v^q_{x,y})$ represents choosing the assignment $v^q_{x,y}$ after the assignment $v^p_{m,n}$. To be consistent with the objective function in Eq. (2.1), we set the weight on the edge $(v^p_{m,n}, v^q_{x,y})$ to be the cost of choosing the decision $v^q_{x,y}$ after $v^p_{m,n}$, *i.e.*, the sum of the

communication latency in transferring intermediate data from p to q, and the computation latency on the node q:

$$c(v^p_{m,n}, v^q_{x,y}) = \sum_{l=x}^{y} \tau^q_l + \epsilon^p_n \tag{2.7}$$

Finally, we add a pair of virtual start/end vertices v^s, v^e connecting to vertices corresponding to the first and last node with zero-cost edges, *i.e.*, $\{v^1_{m,n} | 1 \leq m \leq n \leq L\}$ and $\{v^P_{m,n} | 1 \leq m \leq n \leq L\}$ respectively, to serve as source and destination in the graph.

The optimization objective now becomes finding the shortest path from v^s to v^e, and the vertices along this path form the assignment decision functions u, w, *i.e.*, $u(p) = m, w(p) = n$ if and only if $v^p_{m,n}$ belongs to the shortest path. By reformulating the optimization, we can derive the optimal $u(p), w(p)$ by finding the shortest path with classic linear programming based solutions and avoid the complicated non-standard optimization problem in the original formulation.

2.4.3 Split Cost Information (SCI) design

At first glance, the problem can be straightforwardly solved by applying the well-known Dijkstra's algorithm. However, Dijkstra's algorithm requires reconstructing the entire graph on a central controller node, and the size of the graph grows exponentially with the number of layers and linearly with the number of nodes. In the previous example of ResNet50 running on 5 nodes, the corresponding graph comprises 4.5×10^5 edges, each requiring the profiling of computation and communication latency to determine the cost. Despite the relatively low computation complexity of Dijkstra's algorithm, gathering such information and feeding it back to the controller incurs significant overhead, especially

when the decision needs to be updated frequently under network dynamics.

We now introduce our *Split Cost information (SCI)* design which can efficiently solve the graph representation of the split ML problem. SCI inherits the logic of the distributed Dijkstra's algorithm [57, 58] and is tailored to the split ML graph to tackle the information gathering overhead. In SCI, each vertex calculates its own shortest path by traversing its neighbor vertices' *path cost*, *i.e.* the sum of all edges on the shortest path of a vertex. Specifically, given that a vertex A's neighbor vertices' path costs are known, A selects the neighbor vertex with the minimal sum of shortest path cost and edge cost as its predecessor vertex on the shortest path, and the sum value as its path cost. The shortest path can then be found by iteratively following the predecessor vertex all the way to the destination. Hence, to determine the shortest path, a vertex needs to know the path costs of all its neighboring vertices. However, acquiring such information may induce non-trivial communication overhead. In existing distributed algorithms [57, 58], a vertex needs to send individual messages to all its neighbor vertices with its shortest path cost value. Note that each node has $\binom{L}{2}$ vertices (choosing the start and stop point from L layers for a split assignment) and each vertex has $\frac{(L+1)P}{4}$ outgoing edges on average (a vertex $v^p_{m,n}$ has outgoing links to all the following nodes after p, which is $\frac{P}{2}$ nodes on average, and for each node, the vertex has links to all $L-n+1$ vertices with starting layer n+1, which is $\frac{L+1}{2}$ vertices on average). Therefore, a node needs to send $\binom{L}{2}\frac{(L+1)P}{4} = O(L^3P)$ messages. In the previous example of ResNet-50 running on 5 nodes, this translates to 3×10^6 messages, each with only one cost value. Sending such a large number of short messages all at once incurs huge overhead and can easily congest the network, rendering

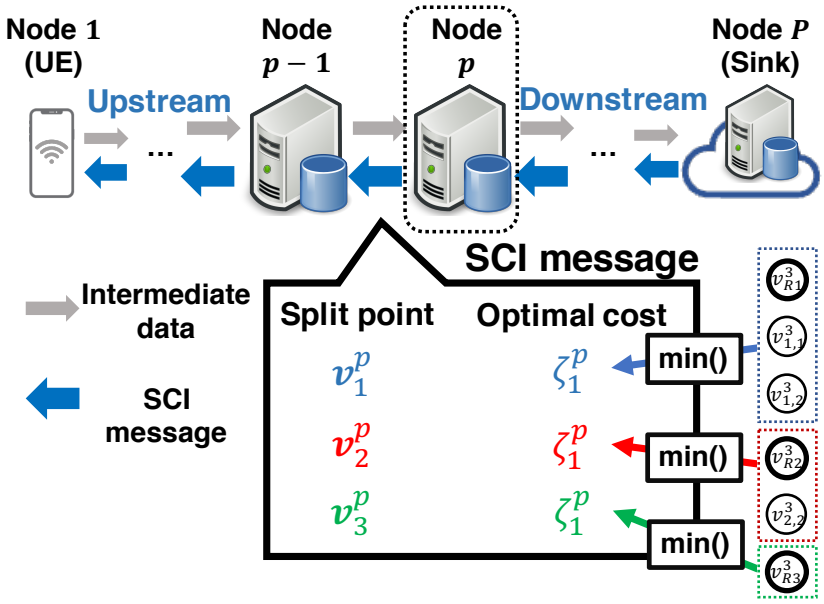


Figure 2.4: A showcase of Split Cost Information (SCI) message transmitted from node p to node $p-1$. The message contains a optimal path cost for each split point set.

it impossible to directly apply the existing algorithm to the split ML problem.

Transforming the split graph. To reduce the communication overhead for distributed shortest path algorithm, we introduce a *split graph transformation* technique. We observe that for a vertex on computing node p, the neighboring vertices are mostly located on the adjacent node $p-1$, except for those vertices that skip the node $p-1$. For example, edge $(v_{1,3}^1, v_{4,6}^3)$ assigns the 6 layers between node 1 and 3 and skips node 2. If we can eliminate such edges, we can limit the communication strictly between two adjacent nodes, and the $O(L^3)$ short messages from a node can be aggregated as one single message. To this end, we transform the graph by introducing *relay vertex* v_{Rn}^P to represent the "null" workload assigned to the skipped node. In the above case, edge $(v_{1,3}^1, v_{4,6}^3)$ can be broken down into two edges $(v_{1,3}^1, v_{R3}^2)$ and $(v_{R3}^2, v_{4,6}^3)$, both connecting the vertices of neighboring

Algorithm 1: Optimal split cost calculation.

Input: Optimal cost for each split point set on node $p+1$:
$\{\zeta_0^{p+1}, \zeta_1^{p+1}, ..., \zeta_{L+1}^{p+1}\}$, layer-wise computation latency:
$\{\tau_0^p = 0, \tau_1^p, ..., \tau_L^p\}$, layer-wise communication latency from node $p-1$:
$\{\epsilon_0^{p-1}, \epsilon_1^{p-1}, ..., \epsilon_L^{p-1}\}$ where ϵ_0^{p-1} corresponds to model input

Output: Optimal cost for each split point set (SCI message) $\{\zeta_0^p, \zeta_1^p, ..., \zeta_{L+1}^p\}$,
Optimal split points $\{n_0^p, n_1^p, ..., n_{L+1}^p\}$

$i \leftarrow 0$;
while $i \leq L$ **do**
    ; `// Iterate split point sets`
    for $j \leftarrow i$ ***to*** L **do**
        ; `// Iterate vertices in a split point set`
        $\theta_j^i \leftarrow \sum_{l=i}^{j} \tau_l^p + \epsilon_i^{p-1} + \zeta_{j+1}^{p+1}$; `// Calculate the cost of` j`-th vertex in` i`-th split point set`
        ;
    $n_i^p \leftarrow \text{argmin}_j(\theta_j^i)$;
    $\zeta_i^p \leftarrow \min_j(\theta_j^i)$;
    if $n_i^p > i+1$ **then**
        for $z \leftarrow i+1$ ***to*** n_i^p **do**
            $n_z^p \leftarrow n_i^p$;
            $\zeta_z^p \leftarrow \zeta_i^p - \sum_{l=i} z\tau_l^P$;
        $i \leftarrow n_i^p$;
    $i \leftarrow i+1$

nodes. Fig. 2.3 showcases the transformation graph of splitting a 2-layer model on P nodes with relay vertices. With this measure, we aggregate the $O(L^3)$ path cost values into one message, thus saving the overhead of sending $O(L^3)$ individual messages.

Slimming the inter-node messages. The use of relay vertex in the graph transformation reduces the messaging overhead. But the size of a message increases to $O(L^3)$ times due to the aggregation and may still incur non-negligible transmission latency when the ML model has a large number of layers. For example, for a ResNet-152, each message would contain 4×10^6 path cost values. Suppose each path cost value is stored in float format, the size of the message would be 80 MB, which is too large for real-time signaling. We thus further reduce the size of each message by pruning the number of path cost values in a message. We first transpose the graph, *i.e.*, reverse the direction of all edges and reverse the role of the start/end nodes, which does not change the optimal shortest path. We then group the vertices $v^p_{m,n}$ on a node p by the starting layer m. We denote such a group of vertices as a *split point set* $\mathbf{v}^p_m$. Fig. 2.3 demonstrates the split point sets on different nodes. We see that in the transposed graph, a vertex's neighbor vertices must belong to the same split point set. This is because according to Eq. (2.6), a vertex's neighbor vertices all share the same starting layer. Recall that in the shortest path algorithm, a vertex only needs to know the minimal path cost among its neighbor vertices in order to calculate its own path cost. This means that only the cost of the optimal vertex in a split point set, *i.e.* the vertex with the minimal cost, is required by the shortest path calculation, and instead of sending cost values for all vertices, a node can just send one per split point set.

Leveraging the above property, we introduce an *optimal split cost algorithm*, as described in Algorithm 1. The algorithm simultaneously reduces the message size and finds the shortest paths for vertices on a node. Line 1 to Line 5 first calculates the path cost of all vertices. Then, based on the aforementioned property, Line 7 finds the optimal path cost ζ_i^p for each split point set. These path cost values are packed into a signaling message called *Split Cost Information (SCI) message*, as shown in Fig. 2.4, and sent to the adjacent node to serve as the input of the optimal split cost algorithm on that node. In the meantime, Line 6 finds the stop layer index n_i^p of the vertices corresponding to the optimal paths, which are later used as the key information for split assignment decision making. A SCI message contains only $L + 1 = O(L)$ shortest path cost values corresponding to L layers in the model plus a relay node layer. For the previous ResNet-152 example, it means a less than 4KB message size, nearly 24000× smaller than the original 80MB message. The optimal split cost algorithm design can thus be safely extended to models with a large number of layers without inducing large overhead. To further improve the efficiency of the algorithm, we observe that if n_i^p for a split point set i is greater than i, then $n_f^p = n_i^p$ for all split point sets $f \leq n_i^p$. This is because the costs θ_j^f on split point sets $f \leq n_i^p$ is just θ_j^i minus a constant $\sum_{e=i+1}^{f} \tau_l^p$ and the optimality of n_p^i holds for these split point sets. Hence, we compare n_i^p with i in line 8 and skip the computation for iteration $i + 1$ to n_i^p if $n_i^p > i + 1$.

SCI protocol in 5G networks. We now introduce how to execute the above SCI solution framework in 5G MEC networks. As illustrated in Fig. 2.5, the operation consists of two processes: (1) *SCI update*, (2) *Split ML task*. The SCI update runs along

the *upstream* direction, *i.e.*, from sink node to source node, whereas the split ML task runs on the *downstream* direction. During the SCI update process, a node calculates the shortest path costs with Algorithm 1 and sends the SCI message to its upstream node. Since the downstream SCI message is required by the algorithm, the SCI update starts from the sink and moves upstream towards the source node. The split ML task starts immediately after the SCI update is completed. During the split ML run-time, each node receives the ML intermediate data from its upstream node, executes the ML model up to a certain split point, and sends the intermediate data to its downstream node. A node chooses its own split point based on the calculation results from the SCI update and the upstream node's split point. Specifically, recall that Algorithm 1 derives the stop layer index of the optimal vertices n_i^p for each split point set. Since the vertices in a split point set represent split assignments with the same starting layer i, then n_i^p is the optimal split point for node p if its upstream node splits at layer $i-1$. Hence, given the upstream node's split layer index x, a node p can easily identify its optimal split point by finding n_{x+1}^p. Note that since the source node always executes the ML model from the first layer, its optimal split point is always n_1^1. A split ML task process finishes when the entire ML model is executed and the output is sent back to the source or the cloud server for further application-specific processing.

Owing to the split graph transformation and SCI message design, the SCI protocol achieves high efficiency in solving the multi-split assignment problem: A node only sends out one SCI message to its adjacent node per SCI update and the message size is only a few KB. Combined with the low complexity of the optimal split cost calculation algorithm

($O(L^2)$), the SCI update process can be completed in an instant. In our experiment, we observe that the average running time for one SCI update is only 27 ms.

The low running time of SCI update is crucial for combating network dynamics. Due to the variation of the wireless channel and background traffic demand, the link throughput between MECs often varies drastically over time. Consequently, the communication cost profiles used by the optimal split cost calculation are likely to expire very quickly. As a result, the optimal split points calculated by an SCI update also expire quickly, leading to a sub-optimal split. A fast SCI update process means the process can update the optimal split points at a fast pace and thus adapt to more severe network dynamics.

2.4.4 Cost analysis

In this section, we provide a simple analysis to examine the advantages of our split ML approach in terms of computation and communication cost. With the assumption that $\tau_l^P \leq \tau_l^p \leq \tau_l^1, \forall l, p$, *i.e.* the cloud server has the minimal layer-wise computation latency and the UE has the maximal, we can easily deduce that the minimal total computation latency is $\sum_{l=1}^{L} \tau_l^P$ and maximal is $\sum_{l=1}^{L} \tau_l^1$. Note that the maximal total computation cost equals the total cost in that case because UE-only model execution does not entail any communication overhead. Since SCI always selects the split decision with the minimal cost, the maximal possible communication cost in a split ML task is $\sum_{l=1}^{L} \tau_l^1 - \sum_{l=1}^{L} \tau_l^P$, *i.e.*, the difference between the maximal and minimal total computation cost. In other words, SCI allows up to $\sum_{l=1}^{L} \tau_l^1 - \sum_{l=1}^{L} \tau_l^P$ of communication before switching from split ML to conventional UE-only ML. This indicates that unlike conventional cloud-based ML

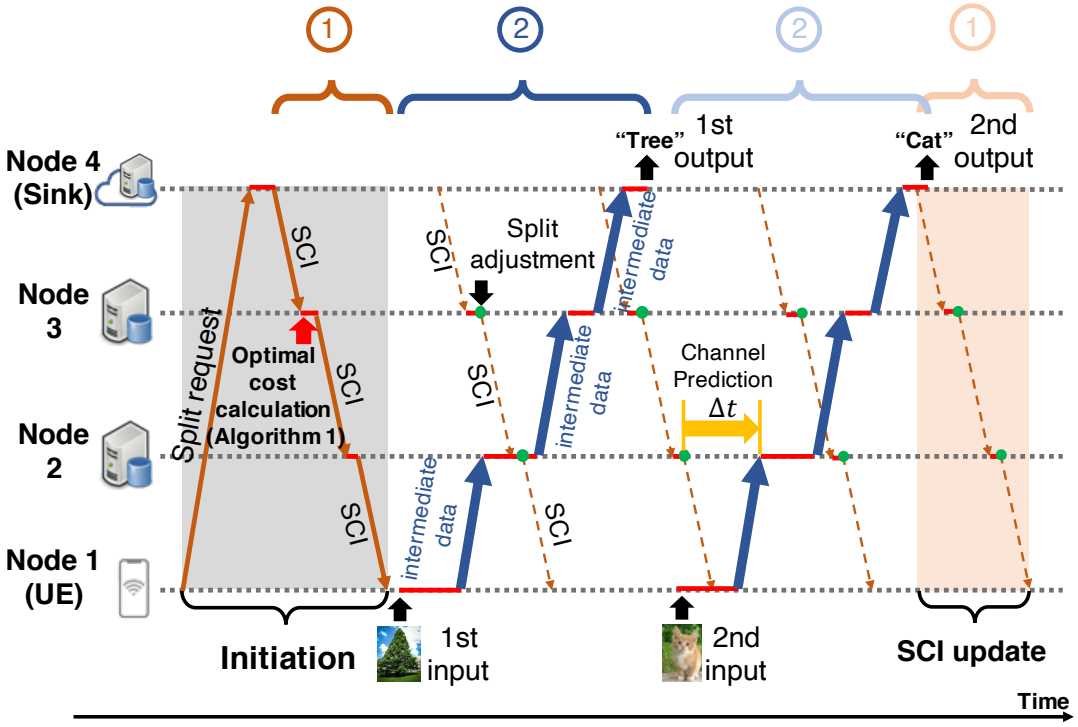


Figure 2.5: Split DNN procedure: ① SCI update: each node calculates its shortest path costs and signal its upstream node with SCI message, ② Split ML task: each node chooses its own split point and execute the layers.

whose communication cost may grow unbounded under poor link conditions, SCI is able to gracefully degrade to the UE-only execution when link capacity becomes too low.

2.4.5 Extension to split DNN training

The above design focuses on splitting a single-pass ML inference model. In contrast, ML training is an iterative bi-directional process: a forward inference pass, same as the ML inference process, is followed by a backward pass, which travels through the layers in reverse order to calculate the parameter updates using intermediate results from the forward pass [41]. The shared intermediate result means *a node needs to have the same set of layers for both the forward and backward passes.* Hence, only one split assignment is needed. Similar to split inference, we can formulate the split assignment as a graph and derive the optimal split assignment by finding the shortest path. However, two additional costs need to be considered in the edge cost of the graph. The first one is the cost of running the backward pass, including both computation and communication cost. The second is the cost of passing the layer parameters when the split assignment changes. To better explain parameter passing, consider a case where two MECs p_1 and p_2 are initially assigned with layer 1-2 and 3-4 respectively. Later the assignment changes to 1-3 for p_1 and 4 for p_2. In this case, the parameters of layer 3 need to be passed from p_2 to p_1. The parameter passing cost only exists in training because in inference the parameters from all layers are fixed and can be loaded to MECs prior to the inference task, while in training the parameters vary rapidly. Assuming the layer-wise training cost $\tau_l'^p$, $\epsilon_l'^p$ and the cost of passing l-th layer's parameter from p to q $\eta_l^{p,q}$ are known, the cost for an edge is the sum

of the forward and backward passes:

$$c(v^p_{m,n}, v^q_{x,y}) = \sum_{l=x}^{y}(\tau^q_l + \tau'^q_l) + \epsilon^p_n + \epsilon'^p_n + \sum_{l\in L_T} \eta^{p,q}_l \tag{2.8}$$

where L_T is a set of the layers required to be transferred from p to q. In the case where the layers needs to be transferred from q to p, we replace $\eta^{p,q}_l$ with $\eta^{q,p}_l$. Therefore, to enable split training, we simply need to modify the corresponding edge cost computation (Line 3) in Algorithm 1:

$$\theta_j \leftarrow \sum_{l=i}^{j}(\tau^p_l + \tau'^p_l) + \epsilon^{p-1}_i + \epsilon'^{p-1}_i + \sum_{l\in L_T} \eta^{p,q}_l + \zeta^{p+1}_{j+1} \tag{2.9}$$

2.4.6 Runtime Optimization under Network Dynamics

The foregoing discussion assumes that the network dynamics can be counteracted by frequent SCI updates for most of time, which is corroborated by our experiments in Sec. 2.7B. However, due to the sparse high variance in 5G links [59], the latency information in the SCI messages may still occasionally expire at the time when the split ML is executed. As a result, the corresponding split decisions become outdated and highly sub-optimal. Furthermore, the latency information in an SCI message is often aggregated across multiple hops before it reaches an upstream node, during which the links may experience larger dynamics. So the optimal decisions made at these upstream MECs are more likely to expire.

To showcase this phenomenon, we create a simplified split DNN scenario with 1 UE and 5 MECs on the route. The link capacities are $\{4000, 2000, 800, 400, 100\}$ Mbps for each hop starting from the UE. We generate dynamic traffic according to the background UE

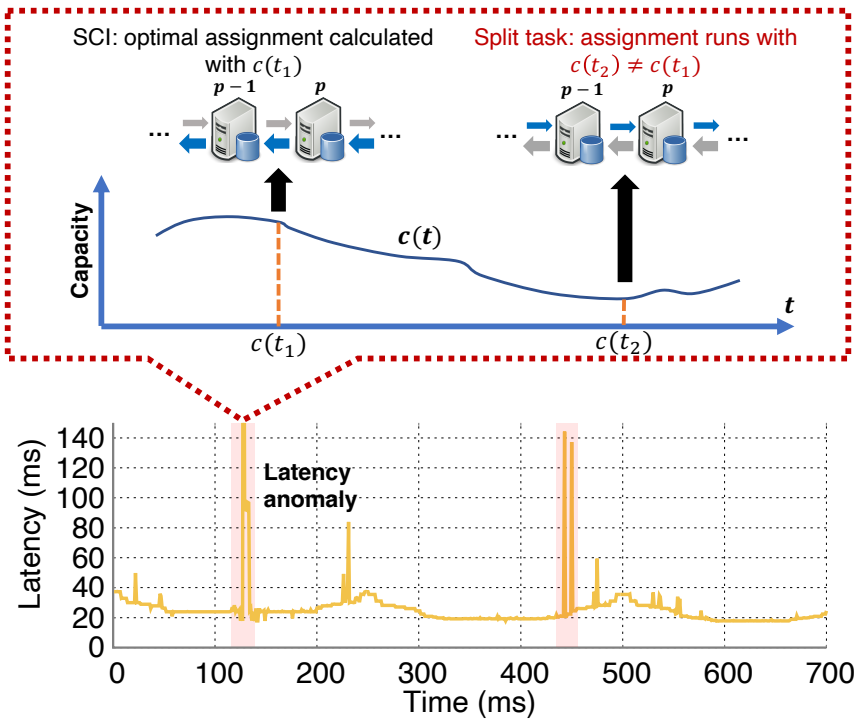


Figure 2.6: Link dynamic showcase: the split ML latency surges at $120ms$ and $450ms$ due to high variances in link capacity.

density and demand distributions extracted from a real-world cellular network trace[60]. Fig. 2.6 shows the time series of total split latency. We see that at $t = 120ms$ and $450ms$, the split latency surges to over 500 ms. As illustrated, this is because the split assignment is made at $t = t_1$ when the access link capacity is $c(t_1)$, while the split task runs at $t = t_2$ when the link capacity drops to $c(t_2)$, making the split assignment outdated.

To tackle the link dynamics, we introduce a simple *predictive splitting mechanism* to the SCI design. This mechanism leverages existing cellular link capacity forecast schemes [61] to predict the inference latency. Recall that in Eq. (2.7), the edge cost involves the communication latency ϵ_n^p, which can be estimated by the link capacity $c^{p,p+1}(t)$ and the

intermediate data size of n-th layer s^n:

$$\epsilon_n^p(t) = \frac{s^n}{c^{p,p+1}(t)} + \psi^n \tag{2.10}$$

where ψ^n is the MEC overhead for transferring n-th layer intermediate data, including the network stack overhead and 5G signaling overhead, which can be profiled in advance. From Fig. 2.6 we see the latency spikes are caused by the link capacity variation between the SCI message at t_1 and the split task at t_2. We denote such time gap as $\Delta t = t_2 - t_1$. Ideally, if Δt is known in advance, a node p at t_1 can forecast the link capacity at t_2 $c^{p,p+1}(t_2) = c^{p,p+1}(t + \Delta t)$ and then calculate the communication latency $\epsilon_n^p(t_2)$ at t_2 from Eq. (2.10), thus eliminating the effect of link dynamics. However, Δt is determined by the dynamics of upstream links and the split assignment of upstream nodes, both of which are not known to node p. Nonetheless, since the upstream nodes only run a part of an ML model, the variation in Δt is largely limited, often smaller than the link coherent time at node p. Hence, we estimate Δt on a node by averaging the time gaps from previous g split tasks on this node and calculate the communication latency accordingly. Note that since procedure 1 and 2 are mirrored, different nodes have different Δt, *e.g.* node 2 in Fig. 2.5 has smaller Δt than node 4. In the case where Δt is smaller than link coherent time, we simply disable predictive split as it is no longer necessary. Experiments in Sec. 2.7 shows the predictive split can eliminate over 95% of latency spikes under realistic cellular link dynamics when the link capacity prediction error is less than 13%.

2.5 HiveMind Multi-objective Split

It is straightforward to apply the HiveMind multi-split to metrics other than latency. One can simply replace the layer-wise latency profiles τ_l^p and ϵ_l^p with the corresponding cost profiles, *e.g.*, the energy consumed for running layer l. However, a 5G MEC application often needs to account for a mix of metrics simultaneously [15, 62]. Depending on the specific application, these metrics may require different objectives. There are two categories of performance objectives defined in 5G [62]: *(i)* Best effort, where a metric needs to be optimized to the best effort, *e.g.*, reducing the energy cost as much as possible, *(ii)* Quality assurance, where a metric needs to be limited by a certain threshold, *e.g.*, making sure the latency is below 100ms. It is non-trivial to apply multiple metrics with different objectives to the shortest path solutions.

The canonical multi-metric shortest path solution linearly combines the metrics as the edge cost in the graph [63]:

$$c(v_{m,n}^p, v_{x,y}^q) = \sum_{j=1}^{M} w_j (\sum_{l=x}^{y} (\tau_j)_l^q + (\epsilon_j)_n^p) \tag{2.11}$$

where $(\tau_j)_l^p$ and $(\mu_j)_l^p$ are the computation and communication costs of the j-th metric, and w_j is the corresponding weight. Such the linear combination method does not require any modification to the HiveMind operations other than the edge cost calculation. However, this method does not distinguish the best effort objectives from quality assurance objectives, and cannot represent the threshold of the quality assurance objectives. Since the objectives of different metrics are often conflicting, *e.g.* optimizing latency may lead to increased power consumption, the linear combination method may lead to excessive op-

timization on the quality assurance objectives and compromise the best effort objectives. For example, in Sec. 2.7D, we run a multi-objective split task with a best effort objective on energy consumption, and a quality assurance objective on latency whose threshold is 120ms. We observe that comparing to the optimal method, the linear combination unnecessarily optimizes the latency to $75ms$ while increasing energy consumption by 29%.

To address the above problem, we introduce a *non-linear weight function design*. The high-level idea is to restrain the weight of quality assurance objectives in the edge cost when the metric is well below the threshold so that the best effort objectives are not affected, and quickly increase the weight of the quality assurance objectives when the metric approaches the threshold to prevent the quality assurance violations. Doing so requires a non-linear mapping between the metric to its weight in the edge cost. Hence, we use non-linear weight functions to reshape the quality assurance metrics, before linearly combining them with the minimization metrics. Suppose there are B objectives in a split ML task, where the first C objectives are quality assurance and the remaining are best effort. We define $\phi_j = \sum_{l=x}^{y}(\tau_j)_l^q + (\epsilon_j)_n^p$ as the cost of the j-th objective on edge $(v_{m,n}^p, v_{x,y}^q)$. Then the total cost of the edge can be calculated as:

$$c(v_{m,n}^p, v_{x,y}^q) = \sum_{j=1}^{C} W_c(\phi_j) + \sum_{j=C+1}^{B} w_b \phi_j \tag{2.12}$$

where W_c is the non-linear weight function and w_b is the linear weight for best effort objectives.

Fig. 2.7 shows the calculation of the edge cost. By design, W_c should be a monotonically increasing convex function that asymptotically approximates the constraint $x = \Phi_j$.

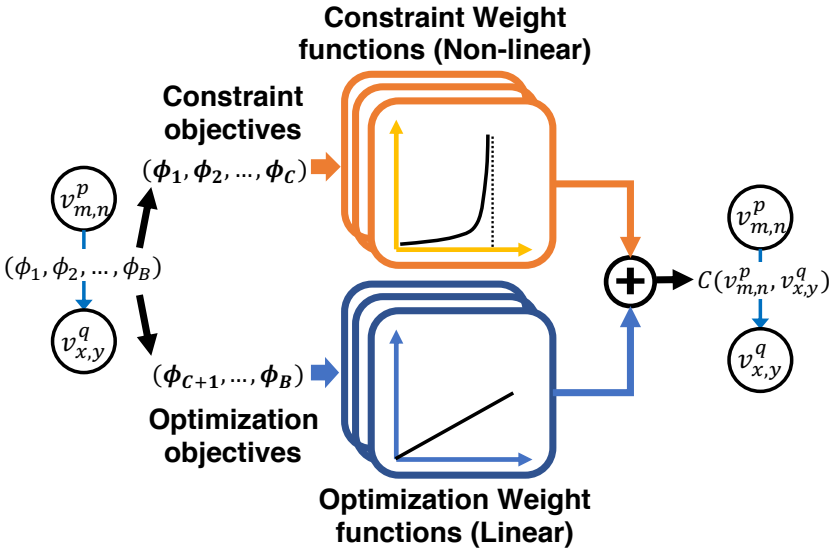


Figure 2.7: The edge cost calculation in HiveMind multi-objective: the quality assurance metrics are reshaped by non-linear weight functions before linearly combined with the best effort metrics.

We empirically find that the inversely proportional function $y = -\frac{1}{x-\Phi_j}, x \in (-\infty, \Phi_j)$ achieves the best optimization performance. Given the updated edge cost definition, we can simply plug it in the optimal latency calculation algorithm (Line 3 in Algorithm 1) to enable the multi-objective split.

2.6 Splitting Non-Linear Neural Networks

The HiveMind design we have introduced so far is applicable to DNNs. In this section, we describe how to extend the design to non-linear NN models, including Recurrent Neural Network (RNN) and collaborative learning models.

2.6.1 Split RNN

RNN models are widely used for applications with temporal correlated and sequential inputs, such as natural language processing and speech recognition. An RNN model consists of a sequence of identical *recurrent modules*. Each recurrent module contains several linearly-organized layers. Part of the model output called "hidden states" is feed to the next module along with the input sequence. Fig. 2.8 (a) demonstrate the topology of a typical RNN model. A straightforward way to adapt HiveMind to RNN is to split the recurrent module since it shares a similar linear structure as a standard DNN. This would require running the recurrent module repeatedly on the MEC network. In the common case where the first layer is on the source node and the final output layer is on a MEC node or cloud server, the hidden states generated at the final layer need to be fed back to the source node everytime the recurrent module is repeated. The size of the hidden states is usually in the same order as the intermediate data passed across adjacent layers, if not larger [64]. Hence, transferring the feedback to the source, usually across multiple hops, causes a large overhead.

To address this problem, we propose to *linearize the RNN splitting problem* so it becomes similar to the DNN splitting. Consider an RNN with R recurrent modules, each consisting of L layers. As shown in Fig. 2.9, we regard the l-th layer on the r-th recurrent module as layer $(r-1) \times L + l$. The remaining input sequence $\{x_{r+1}, x_{r+2}, ..., x_{P-1}\}$ and the previous output sequence $\{y_1, y_2, ..., y_{r-1}\}$ are transferred along with the intermediate output. This way we can reuse the HiveMind mechanism while avoiding sending the hidden

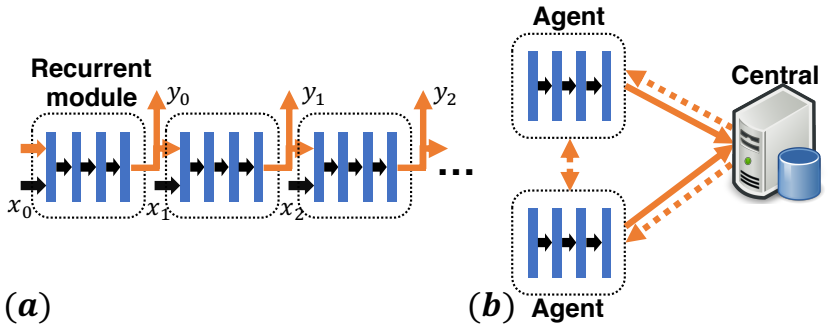


Figure 2.8: Non-linear NN showcase: (a) Recurrent Neural Network (RNN), (b) Collaborative learning.

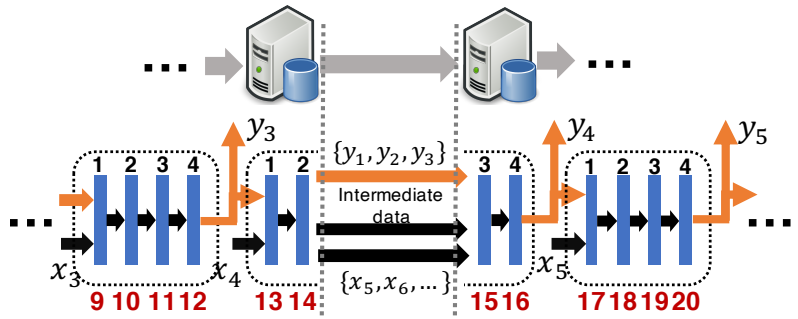


Figure 2.9: A showcase of linearized RNN in HiveMind split RNN design.

state across multiple hops within the MEC network.

2.6.2 Split Collaborative Learning

Collaborative learning is a widely adopted distributed ML training paradigm. Common collaborative learning models include Distributed Deep learning (DDL), Multi-agent Reinforcement Learning (MARL), and Federated Learning (FL). In a collaborative learning setup, multiple identical copies of a model are deployed on different nodes called *agents*, each training its models with locally observed environment and states. In order to achieve global optimum, the agents periodically transfer their model parameters to each other

or to a central controller to merge the parameters, as shown in Fig. 2.8(b). The model parameters are usually on the order of 100 MB [65], significantly larger than inter-layer intermediate data which are on the order of 100 KB (Sec. 2.7B). As a result, the parameter transfer cost constitutes a significant part of collaborative learning.

To optimize the runtime cost of an agent network, the HiveMind design needs to account for the cost of parameter transfer. The parameter transfer happens after calculating the final layer of an agent model. Hence for split assignments on node p that involve the final layer, *i.e.*, $v^p_{i,L}, 0 \leq i \leq L-1$, we need to add the parameter transfer cost to the edge costs. Suppose the parameter transfer cost at node p ω^p is known through link capacity profiling. Then, we can simply add ω^p to the standard HiveMind training (Eq. 2.9) to reflect the parameter transfer cost:

$$\theta_j \leftarrow \sum_{l=i}^{j}(\tau^p_l + \tau'^p_l) + \epsilon^{p-1}_i + \epsilon'^{p-1}_i + \sum_{l \in L_T} \eta^{p,q}_l + \zeta^{p+1}_{L+1} + \omega^p \tag{2.13}$$

We can then replace the cost calculation for $v^p_{i,L}, 0 \leq i \leq L-1$, *i.e.*, the L-th iteration of the inner for-loop in Algorithm 1, with Eq. (2.13) to enable split collaborative learning. Note that since the split assignments that do not involve the final layer are not affected by the parameter transfer cost, the cost calculations in the first $L-1$ iterations remain unmodified.

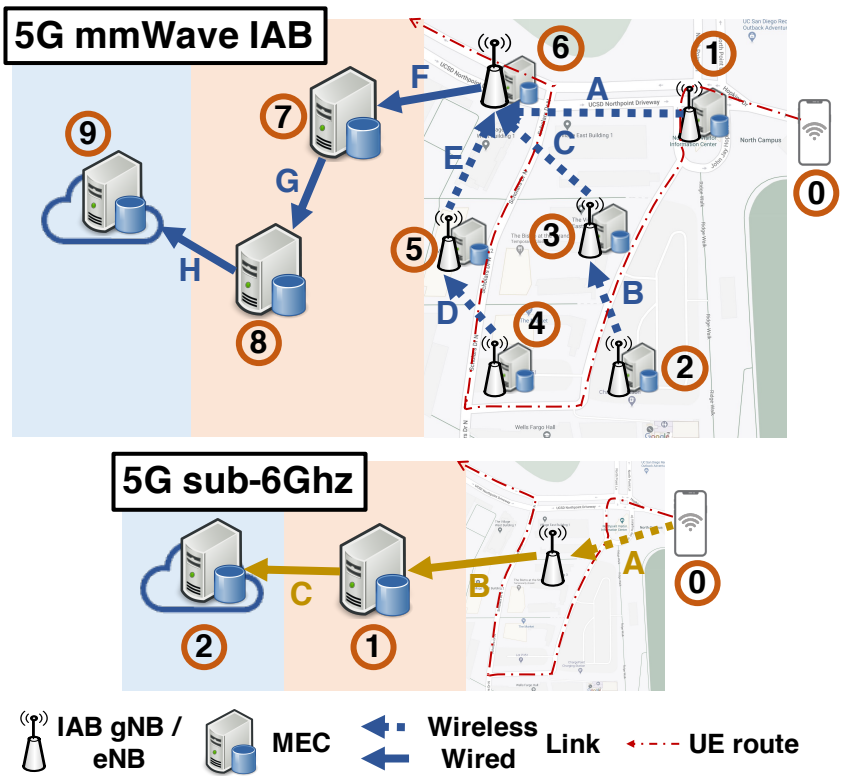


Figure 2.10: Simulated 5G network topology and UE trajectory.

2.7 Evaluation

2.7.1 Simulation setup

We evaluate the HiveMind framework on a custom built 5G network simulator. Below we introduce the key components of the simulator.

Network topology. We evaluate HiveMind on 2 representative 5G networks: a 5G mmWave IAB network and a 5G sub-6 GHz network. As illustrated in Fig. 2.10, the 5G mmWave IAB network consists of 6 IAB nodes, each equipped with a mobile MEC. The IAB nodes are deployed in a $1km \times 1km$ region, and form a tree-topology in compliance with the 3GPP Release-15 guidelines [30]. Traffic from the IAB nodes is aggregated and flows to the core network through the donor IAB (node 6) and its wired backhaul link (link F). There are 2 additional MECs (node 7 and 8) in the core network. A cloud server is connected to node 8 through public IP, serving as the sink node. The sub-6 GHz network consists of a single basestation. There is one MEC in the core network and a cloud server in public IP. All wireless link capacities are configured according to the maximum uplink MCS' bitrate specified by 3GPP [66], as shown in Table 2.1.

UE mobility. A target UE within the IAB network's coverage area requests the split ML task at the beginning of each experimental trial. It keeps running the task while moving along a pre-defined trajectory at a speed of $40km/h$. We calculate the UE's access link SNR using the Friis model assuming an EIRP limit of 40dBm as specified by the FCC [67]. We then convert the SNR to link bitrate by 5G NR CQI to MCS mapping table [68]. Although this channel and link model does not account for sophisticated propagation

effects, it should suffice to generate a similar level of dynamics as real-world mobile cellular links, which is critical for testing the model splitting.

Background traffic. We generate the number of UEs and individual UEs' bit-rate distribution following the traffic emulation approach in [60], which has been cross-validated with real-world traffic traces. We then feed the background UEs' bit-rate samples along with the target UE's bit-rate to a widely-adopted Proportional Fair Scheduler (PFS) [69], which allocates the channel resources and determines the link capacity for the target UE. In addition, we scale the UE population and individual UEs' bitrate by $2, 1$, and $\frac{1}{2}$ to create high, mid, and low background traffic scenarios.

ML cost profiles. The computation and communication cost profiles of an ML model are crucial for a faithful representation of the model workload, and also serve as inputs to the HiveMind framework. To obtain realistic ML cost profiles, we build a Python-based latency/energy profiling tool as a stand-alone package for Pytorch. The profiling tool traverses the ML model object and registers a forward hook and a backward hook for each layer (a `torch.nn.functional` object). To generate latency profiles, the hooks measure the processing time of a layer during inference and training phases using Python's built-in `time.perf_counter()` module with an accuracy of $\pm 1\mu s$. The per-layer processing time values are organized in the order of the layers as the computation latency profile. The hooks also record the output size of the layer for inference and training, which is used to calculate the communication latency based on Eq. (2.10). For computing energy profiles, we adopt *PyJoules* [70], a third party energy footprint monitor that measures the computing energy of a Python function. For communication energy profiles, due to the

lack of 5G interface on our devices, we adopt the link bitrate/power mapping data from the latest operational 5G measurement [59] to convert the instantaneous link bitrate in our simulation to power consumption. We measure the cost profiles on the following four sets of machines to represent the UE, IAB MEC, core MEC, and cloud server, respectively: *(i)* Raspberry Pi 3 Model B+, *(ii)* MacBook Air 2020 with Apple M1 CPU and 16GB unified memory, *(iii)* A PC with Ryzen 3800X CPU, 64GB DDR 4 RAM, and Nvidia RTX 2080 GPU *(iv)* A server with Intel 9990XE CPU, 128GB DDR4 RAM, and 4 $\times$ Nvidia RTX 1080Ti GPUs.

2.7.2 Multi-split performance validation.

Impact of background traffic To evaluate the effectiveness of HiveMind under different background network traffic intensities, we run HiveMind inference and training along with three baseline approaches: *UE Only*, which runs the whole model on the UE; *Cloud Only*, which runs the whole model on the cloud server; *Single Split*, which splits the network only between the UE and a cloud server as in [16]. We use ResNet18, a widely used image classification CNN with a moderate number of layers (52) as the split ML model. For inference tasks, we choose the total processing latency as the optimization objective. For training tasks, we choose the sum energy consumption of IAB MECs and UE as the optimization objective, since the training tasks are not as latency-sensitive as the inference tasks.

Fig. 2.11 and 2.12 plot the running cost distribution of each method under the mmWave IAB and sub-6GHz networks, respectively. We see that for both mmWave IAB

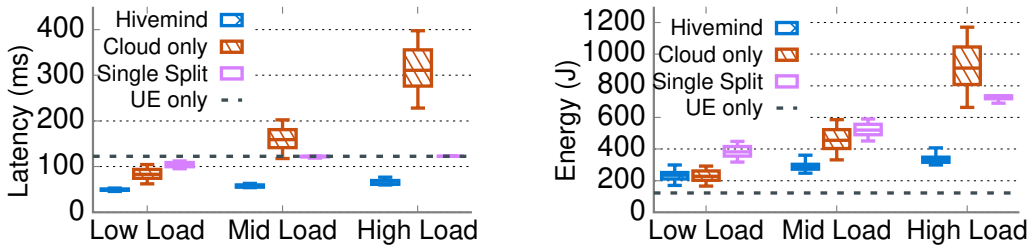
Table 2.1: Link settings

Index	Type	Capacity (Mbps)	UE bandwidth	
			Mean	Std.
5G mmWave IAB				
A	Wireless	200	66.49	19.47
B	Wireless	200	71.89	19.72
C	Wireless	200	47.06	8.95
D	Wireless	200	54.03	11.61
E	Wireless	200	48.59	9.41
F	Wired	400	54.05	12.94
G	Wired	200	36.03	8.63
H	Wired	50	18.12	4.43
5G sub-6GHz				
A & B	Wireless	50	33.94	8.45
C	Wired	50	18.78	3.66

and sub-6GHz networks, HiveMind achieves the minimal running cost for all traffic intensities. For mmWave IAB split inference (Fig. 2.11(a)), the latency of the cloud-only baseline quickly increases to over 300ms as the network becomes more loaded, while the single split converges to the UE-only baseline due to the increasing communication latency. HiveMind is able to keep a stable running latency of $\leq 79\ ms$ by limiting the usage of the core and cloud servers under high network load. For mmWave IAB split training (Fig. 2.11(b)), the energy consumption of the cloud-only method reaches up to $890J$ due to the less power efficient low bitrate links under high network load. HiveMind keeps the IAB nodes' energy consumption under $400J$ by adjusting the split to restrain the data transfer on the links. This set of experiments proves that in the mmWave IAB network, HiveMind can adapt the split assignment to the dynamic network load and outperforms all the baselines on both the inference and training tasks.

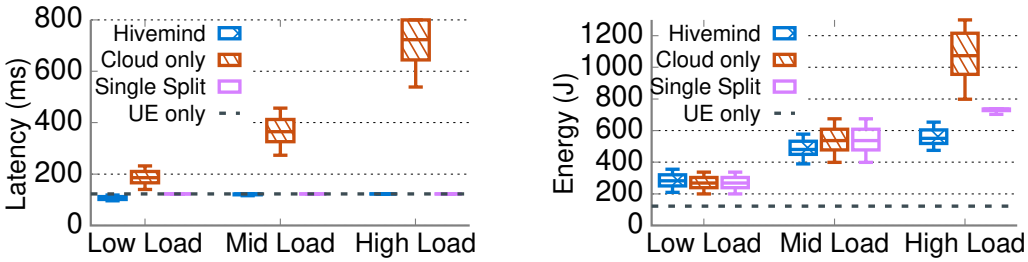
For sub-6 GHz split inference, we observe that the running latency of HiveMind converges to the UE-only approach as the network load increases. The underlying reason is that, due to the relatively lower link capacity (high communication latency) outweighs the low computation latency on the MECs, so that the best split strategy is simply to run the whole model locally on the UE. The result verifies that, even when the network condition provides no benefit for splitting the ML, HiveMind is able to converge to the best non-split strategy and yield the best performance.

Impact of ML models. We repeat the above experiments with three ML models with different computation and communication characteristics: *VGG11*, a 16-layer CNN model with an average layer output size of 356KB; *ResNet18*, a 52-layer CNN model



(a) (b)

Figure 2.11: Efficiency of (a) HiveMind split inference and (b) HiveMind split training, in mmWave IAB network.



(a) (b)

Figure 2.12: Efficiency of (a) HiveMind split inference and (b) HiveMind split training, in sub-6GHz network.

with an average layer output size of 80KB; *R-CNN*, a two-part CNN with the average layer output size of 242KB. We define *relative efficiency* of a strategy as the average ratio between the cost of the strategy and the cost of the UE-only baseline under the same scenario for both training and inference. Fig. 2.13 shows the results. We see that HiveMind achieves the highest relative efficiency on RCNN, 3× over ResNet18 and 1.8× over VGG11. This is because the large communication overhead corresponding to the

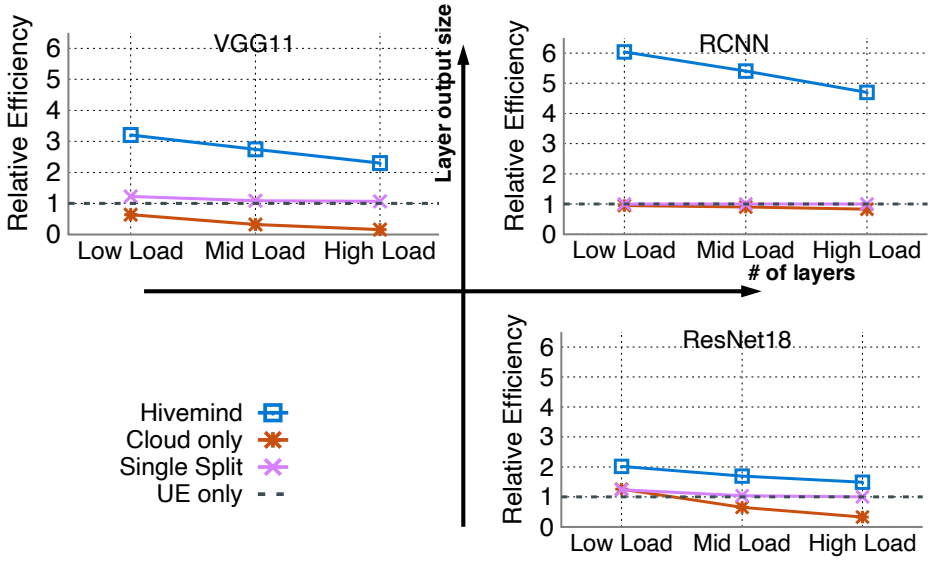


Figure 2.13: Impact of ML models on HiveMind.

larger layer output size provides HiveMind with more improvement margin, while a large number of layers gives HiveMind more flexibility to split the model in more efficient ways. In other words, HiveMind sees more performance gains with a larger layer output size and a larger number of layers.

Impact of MEC capability. In this section we examine the performance of split ML with different MEC capabilites. The capability changes on different MEC nodes may affect the split ML differently depending on the MEC servers' proximity to the UE. Therefore, we isolate the impact of capability change for each type of MEC. Specifically, we repeat the above ResNet18 inference experiment and in each trial, choose one type of MEC nodes and scale their computation latency profiles to $\{\times 1/10, \times 1/2, \times 1, \times 2, \times 10, \times 20\}$, in order to represent the different computational capabilities while keeping the rest of the MEC nodes' latency profiles unchanged. The results are denoted as *improved IAB*, *improved CN*, and *improved cloud* for IAB MEC nodes, core MEC nodes, and the cloud

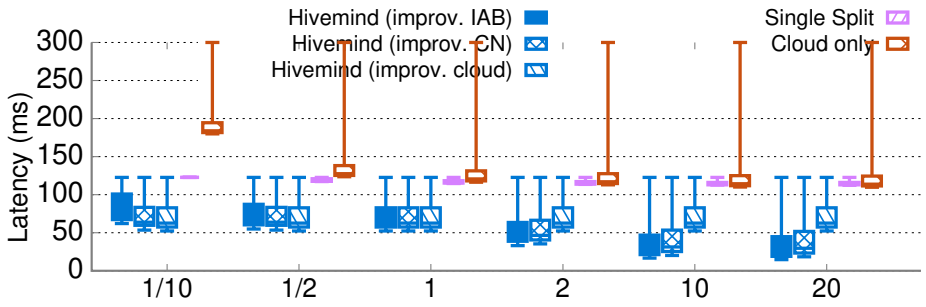


Figure 2.14: Impact of computation capability of various MECs on HiveMind split inference.

server, respectively. Note that for cloud only and single split baselines, we only evaluate the performance under cloud server capability change since they do not use IAB or core MEC nodes. Fig. 2.14 shows the latency under different MEC capabilities. We see that as the scale increase from 1/10 to 20, the improved IAB MECs result in the most latency reduction, 14.5% lower latency than the CN MECs, and 47.2% than the cloud server. This implies that *the split DNN benefits the most with computational capacity improvement on IAB MECs, which are closest to the UEs.* Meanwhile, the single-split and cloud-only approaches converge to 115 ms and 181 ms regardless of the MEC capabilities, indicating the bottleneck of these two approaches lies in the communication latency. We also observe that the overall latency performance for $\times 10$ and $\times 20$ is very close, which shows that improving MEC capability cannot improve the split ML's performance indefinitely since it is also bounded by the communication links' capacity.

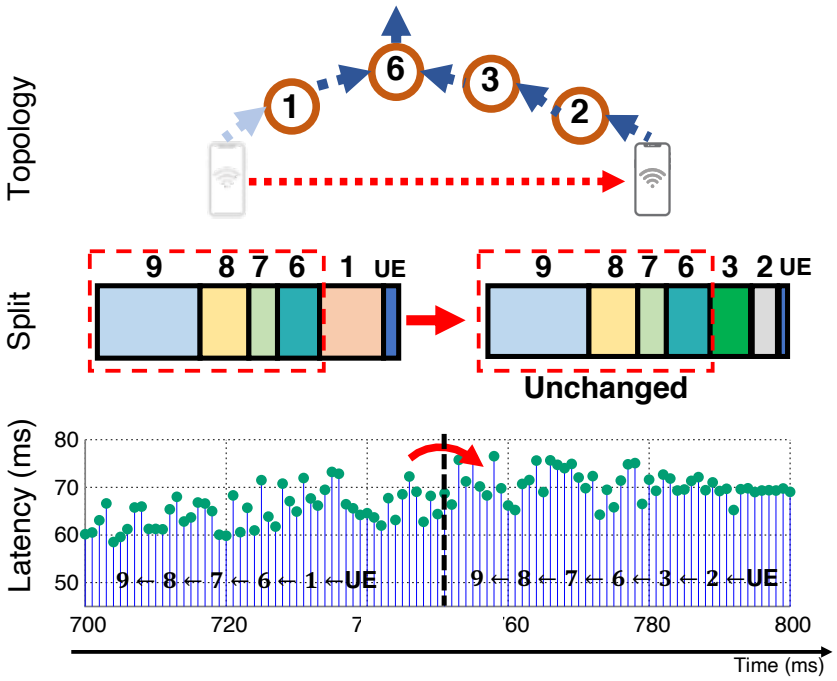


Figure 2.15: HiveMind topology adaptation showcase: the split assignment does not change for the unchanged part of the route and the average latency increases less than 5ms after the topology change.

2.7.3 Performance under network dynamics

Impact of network topology changes. In 5G IAB networks, the network topology may vary over time, as some gNBs may be put into sleep, or additional gNBs are added for load balancing purposes. Meanwhile, the UE mobility and handoff also causes changes in the network path. To showcase the effectiveness of our HiveMind design under such topology dynamics, we randomly select a a period of time in the above ResNet18 experiment where a topology change happens due to UE mobility, and plot its corresponding workload and latency change in Fig. 2.15. This topology change represents an extreme case where not only the access gNB changes from gNB1 to gNB2 due to the UE mobility

but also the number of hops on the route increases by 1. After the topology change, we see the workload assignment remains the same for the unchanged part of the route. Such persistence originates from a unique property of our HiveMind design, *i.e.*, *a MEC node's optimal split decision is independent of the upstream MECs* (Sec. 2.4). Moreover, *HiveMind is stateless and does not require data transfer between the old and new MECs upon a gNB handoff*, which further improves the responsiveness of the system in the presence of UE mobility. We also see a less than 5 ms increase in average latency after the topology change event. This implies the added MEC can compensate for the extra communication latency caused by the extended route and vice versa.

Effectiveness of predictive split. To investigate the performance of HiveMind's predictive split under different levels of network dynamics, we compare it with the standard split DNN. We use link coherent time, *i.e.*, the time during which the link capacity is stable, to represent varying levels of network dynamics. We choose 10ms coherence time to represent extreme network dynamics and 50ms coherent time to represent typical network dynamics in cellular networks during rush hours [71]. We compare the predictive split with the standard split DNN Fig. 2.16 shows the result. We see under 10 ms link coherent time, the predictive split still leaves 12% of latency spikes over 100ms. As mentioned in Sec. 2.4E, this is because the time gap Δt between the SCI message and the split task is much larger than the link coherent time. With the same amount of time gap estimation error, the link capacity estimation is more likely to deviate from the correct value under short link coherent time. In contrast, predictive split eliminates all latency spikes over 100ms when link coherent time is 50 ms, which is similar to the average look-ahead time

for predictive split in our setup. This implies *the predictive split mechanism is effective, as long as the link coherent time is no less than the average look-ahead time of the split decision.*

Impact of link prediction accuracy. The previous experiment assumes a 100% link prediction accuracy. We further investigate the impact of link prediction errors on HiveMind's performance. Reusing the experiment setup for the 50ms link coherent time, we run the split task with different link prediction accuracy. We model the link prediction error as a normal distribution with the correct link capacity as mean and various relative prediction errors as standard deviation. We generate random samples from these distributions for predictive split. Fig. 2.17 shows the median latency against relative prediction error. We see the latency performance deteriorates when the prediction error increases and the performance gain turns to loss at 13%. Given that the state-of-art link prediction mechanism can achieve $< 10\%$ prediction error over 91% of time [61], the result implies the *predictive split is able to tolerate typical link prediction errors.* Note that the median latency converges to 130 ms as prediction error increases beyond 17%, because as the prediction error increases, it is more likely that the access link or one of the IAB links is predicted with extremely small capacity and the predictive split assigns the whole model to UE and IAB MECs, which takes around 130 ms to run the model.

2.7.4 Effectiveness of multi-objective split

To evaluate the effectiveness of the multi-objective split, we repeat the above ResNet18 experiment under intermediate network load. The objective is to minimize

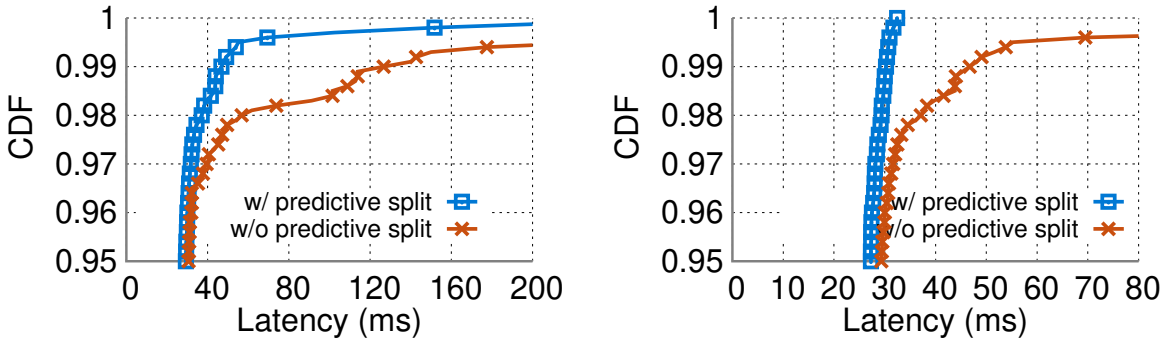


Figure 2.16: Predictive Splitting performance under (a) 10ms and (b) 50ms link coherent time

energy consumption (best effort) while limiting the latency below 120 ms (quality assurance). We compare 's multi-objective design with three baselines: *Opt. latency* which only optimizes latency, *Opt. energy* which only optimizes energy, and *linear combine*, which linearly combines the objectives to calculate the edge cost as discussed in Sec. 2.5. Fig. 2.18 shows the latency and energy consumption distribution. We see that both Opt. latency and Opt. energy achieve the overall minimal for their targeted metric, while sacrificing the other metric, implying that energy and latency are two conflicting objectives in such split tasks. Linear combine balances the two metrics and achieves an average cost of $75ms/476J$, whose latency value is unnecessarily low. In contrast, the HiveMind multi-objective design further reduces the average energy consumption by 22.9% on top of linear combine while keeping the maximal latency under the $120ms$ constraint. The result indicates that HiveMind multi-objective design is able to simultaneously accommodate the best effort and quality assurance objectives.

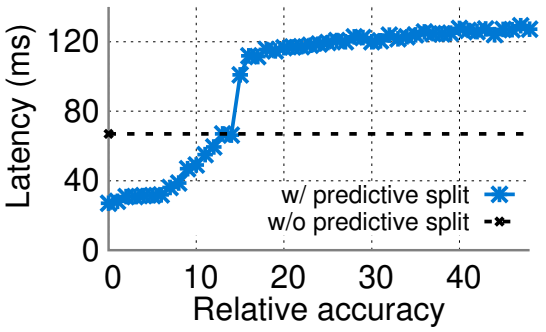


Figure 2.17: Impact of link prediction accuracy on the predictive split.

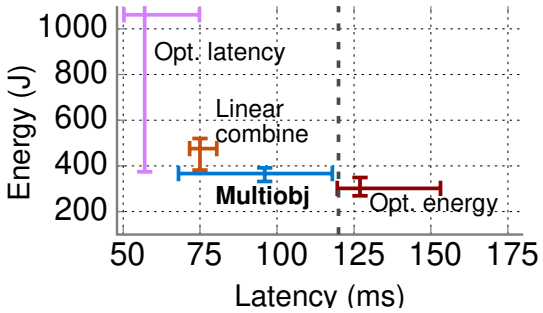
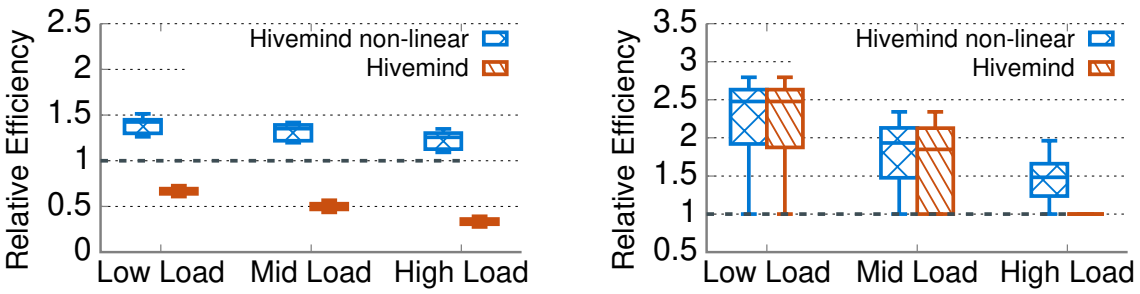


Figure 2.18: Energy consumption and running latency comparison of HiveMind multi-objective and baselines.

2.7.5 Effectiveness in splitting non-linear ML models

We investigate the performance of split collaborative learning and split RNN under varying traffic loads in the mmWave IAB network. To evaluate the split RNN model, we build a customized sequence labeling RNN consisting of 200 GRU recurrent module. For the split collaborative learning model, we choose a well known multi-agent reinforcement learning model, QMIX [72], which consists of a central mixing model and N RNN-based agent models. We assume the central model is located at the cloud server (the sink) and the optimization only applies to the agent models. For both experiments, we set the energy consumption as the optimization objective with a latency constraint of $120ms$. We compare split RNN with the standard HiveMind design which repeats a split recurrent module every iteration and split collaborative learning with the standard HiveMind which splits the agent network without considering the parameter transfer cost. Fig. 2.8 shows the relative efficiency. We see the split RNN outperforms the standard HiveMind by 1.7×



(a) (b)

Figure 2.19: Efficiency improvement of HiveMind non-linear on (a) RNN model (GRU) and (b) collaborative learning model (QMIX) over standard HiveMind split.

under low load and 2.3× under high load, due to the increasing hidden state transfer cost under the high network load. Similarly, split collaborative learning outperforms standard HiveMind by 1.5× under high network load, where the latter fails to account for the large parameter transfer cost. The results prove that *the split non-linear designs can capture the unique cost components in non-linear NNs and outperform the standard HiveMind by a large margin.*

2.8 Conclusion

In this paper, we have explored the multi-split ML as a new paradigm to integrate edge intelligence to 5G systems. Our HiveMind framework distributively finds the optimal multi-split under network dynamics and adapts to multiple optimization objectives and neural network structures. Our experiments demonstrate that HiveMind significantly improves ML running efficiency with various network dynamics, ML models, and MEC

capabilities. We believe HiveMind envisions a new direction to harness collaborative edge power to boost ML intelligence in the 5G era.

Chapter 2 contains material from "HiveMind: Towards Cellular Native Machine Learning Model Splitting" by Song Wang, Xinyu Zhang, Hiromasa Uchiyama, and Hiroki Matsuda, which appears in the IEEE Journal on Selected Areas in Communications, 40(2):626–640, 2021. The dissertation author was the primary investigator and author of this paper.

Chapter 3

Error Tolerant ML Model Splitting Over Edge Networks

3.1 Introduction

Owing to the standardization of 5G multi-access edge computing (MEC) [15] and booming development of machine learning (ML) applications, we have witnessed a growing interest in synergizing the user equipment (UE), MEC, and the cloud to boost mobile ML in recent years [73, 12, 74]. The combined computation power from many devices and the flexibility of splitting workloads enable this new distributed ML paradigm to break the limits of scarce computation resource and unpredictable latency, which used to hinder the UE and cloud based execution, respectively. Among the many proposed solutions of distributed ML, two have recently gained major traction: *Split ML* [16, 40, 41, 29] dynamically assigns parts of a model *inference* process to different computing nodes based

on network conditions and computation resources, to alleviate the pressure of computation on UE devices and potentially optimize end-to-end latency and energy consumption [12, 13, 28, 16, 29]. *Federated Learning (FL)*, on the other hand, distributes the *training* of a model to a federation of participant devices. Each device (usually a UE) trains a copy of the model with its local data, and updates the learned weights to a central parameter server (PS) for aggregation. An FL system utilizes the computation power of a large number of mobile devices, while eliminating the need for uploading privacy-sensitive raw data to the cloud [35, 38].

Despite the unprecedented computation power scale-up, these distributed ML methods suffer from one major bottleneck: due to their distributed architecture, computation nodes need to exchange data with each other (referred to as *ML data*) intensively, which may cause significant communication overhead. Specifically, *split ML transmits intermediate layers' output data from the UEs to MECs, and FL's participating devices upload the locally trained model weights to the parameter server.* Since the data source is usually located at the UE, split ML and FL mainly rely on the cellular uplink for data transfer. Due to the limited power budget of mobile devices, the uplink typically experiences lower link quality [75]. As illustrated in Fig. 3.1, under poor link conditions, block errors occur more frequently and the cellular link needs to attempt multiple retransmissions to correct the errors. This tends to hamper latency sensitive ML applications such as remote driving, remote-controlled robotics, and AR display/gaming which require a stringent end-to-end latency of around 10 ms [11].

Existing works have approached the communication overhead problem by com-

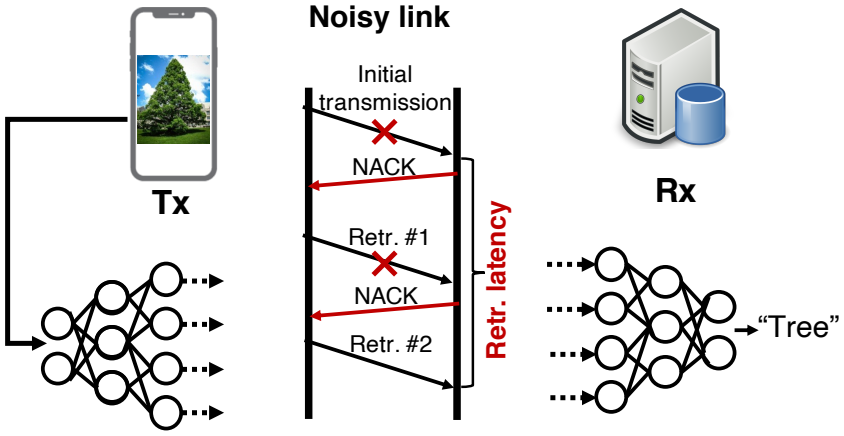


Figure 3.1: An example of split ML over a lossy edge network.

pressing the ML data before transmission [56, 51]. Such approaches often need non-trivial redesign on the application level, and are unaware of the low-layer overhead due to poor link quality. In this paper, we propose to leverage the intrinsic error-tolerant capability of the ML data to circumvent the communication overhead. Unlike the general cellular network traffic that requires error-free data transmission, we observe that distributed ML may still perform the inference or training operations with fair accuracy, even with the present of errors in data transfer. As a result, retransmission becomes unnecessary as long as the error-induced accuracy loss is tolerable. In other words, such error tolerance capability in ML data reduces the need for retransmissions and consequently, the overall communication overhead.

To realize this principle, we present a first study on the error-tolerant capability of distributed ML models. By examining representative split ML and FL model execution over 5G edge networks, we address the following key questions:

How well can the ML data tolerate errors? We perform a layer-level char-

acterization of the accuracy performance versus error rate under a wide range of link conditions and block error patterns. Our experiments reveal that the error tolerance capability exists in most of the commonly used deep neural network (DNN) layers in split ML and FL. However, the specific accuracy losses vary significantly across different types of DNN layers. For a certain layer, the accuracy loss and error rate are strongly correlated, and hence the accuracy performance of the ML model can be profiled as a function of error rate. In addition, the ML data in split ML suffer more accuracy loss from burst errors, whereas FL is agnostic to error patterns.

How to enhance the error-tolerant capability of ML models? Based on the error-tolerance characterization, we propose two techniques to enhance error tolerance in distributed ML: *Interleaved coding* utilizes the property that split ML data is prone to burst errors. It randomly interleaves the ML data sequence so that the burst errors are jumbled into smaller pieces and spread over time, which mitigates their impacts on the model accuracy. *Importance-based coding* facilitates Unequal Error Protection (UEP) on parameters with different values and reduces the error rate on high-valued parameters which are more likely to impact the model accuracy.

How to leverage the error-tolerant capability to improve the efficiency of distributed ML in wireless edge networks? We propose NeuroMessenger, a lightweight cellular-native mechanism that reduces the latency of distributed ML over edge networks. NeuroMessenger is jointly executed by the transmitter and receiver of the ML data, *i.e.* UE and basestation co-located with MEC. It is transparent to ML applications and tightly integrated with the devices' protocols stack which aligns with 3GPP's

vision for future edge intelligence system [11]. For an ML model, NeuroMessenger performs a layer-wise profiling of the error rate to accuracy mapping offline. At runtime, NeuroMessenger on transmitter first reduces the ML data size by pruning the redundant parameters. Then it enhances the error tolerance of the ML data by applying the aforementioned enhancement schemes. The encoded data is then sent over the cellular edge link. At the receiver end, NeuroMessenger's retransmission controller retrieves an estimation of the channel state and estimates the corresponding error rate. Then, based on the error rate to accuracy profiles from the offline stage, the retransmission controller predicts if the accuracy can meet the application requirement and determines if a retransmission is necessary and how aggressive the retransmission should be. By shrinking the need for retransmission, NeuroMessenger substantially improves the communication efficiency of distributed ML while maintaining a user-defined accuracy requirement. With NeuroMessenger, the edge link can aggressively choose a high order modulation and coding scheme, which leads to high raw bit-rate and high block error rate under moderate or low channel quality. NeuroMessenger also greatly reduces the complexity of distributed ML system's development and deployment by making the communication overhead reduction mechanism transparent to applications.

We evaluate NeuroMessenger on a 5G NR simulator. Our experiments adopt the typical PHY settings of a 5G uplink (*e.g.*, modulation and coding scheme, bandwidth and subcarrier spacing, transport block size, *etc.*). We choose state-of-art image classification and speech recognition models as representative ML applications. Our experiment results show: (i) NeuroMessenger reduces the communication latency by up to 95% comparing to

baselines while still maintaining less than 10% accuracy loss under an extreme link SNR of -2 dB. (ii) NeuroMessenger is effective under a wide range of error rates. We observe 20% to 99% latency reduction and less than 5% accuracy loss under 0.1 to 0.95 block error rate. (iii) For split ML models, NeuroMessenger is effective regardless of the partitioning point within the models.

The major contributions of this paper are as follow: (i) A First characterization of error tolerant capability in distributed ML. (ii) A novel system that enhances and utilizes error tolerant capability to reduce communication overhead in distributed ML. (iii) Experimental verification of NeuroMessenger on a 5G edge network environment with representative distributed ML settings.

3.2 Related Work

3.2.1 Distributed Edge Intelligence

Distributed ML on cellular edge servers has garnered much interest in the past two years. In particular, the 3GPP standardization group has envisioned a deep integration of split ML and FL for mobile inference and training respectively [30]. Existing research on split ML mostly focused on partitioning a DNN's computing load to two or more parts which are executed by generic edge servers to meet latency or energy optimization objectives [16, 40, 41]. The research on FL proposed new designs on aggregation algorithm [38, 76, 77], participants selection [78, 79, 80], and incentive mechanisms [81, 82] to improve the communication efficiency or privacy. These works assume the ML data is transmitted

directly over the communication links which usually over-protect the data integrity. When the link condition is poor, frequent retransmissions may occur. *Even under good link conditions*, the communication PHY layer conservatively chooses the modulation and coding scheme (MCS) that is most likely to work in an error-free manner, rather than choosing one with high raw bit-rate but more block errors. In contrast, we propose to build error tolerant capability into the intermediate data, so that they can be salvaged in spite of errors. With this measure, the edge network can avoid the costly retransmissions, and can aggressively choose a high but error-prone MCS level to improve communication efficiency.

3.2.2 ML communication overhead reduction

A classical approach to reduce the communication overhead in edge ML is to compress the ML data. Yao *et al.* [55] proposed a NN-based compressor/decompressor design that compresses the intermediate data following compressive sensing theory. Hu *et al.* [83] adopted a similar NN-based compressor/decompressor and leveraged the prior transmissions to aid the decompression of current intermediate data. These NN-based compression techniques greatly reduce the intermediate data size, but they bear two limitations. First, the NN compressor/decompressor themselves require hours of training for individual model and split point [55]. In the case where the split point or the ML model needs to be constantly changed due to network dynamics, *e.g.*, in the context of online learning, it is impossible to train the compressor/decompressor in real time. Second, such compression approaches are heavily customized to individual application, *e.g.*, the compressor/decompressor are jointly trained with the ML model. In contrast, NeuroMessenger

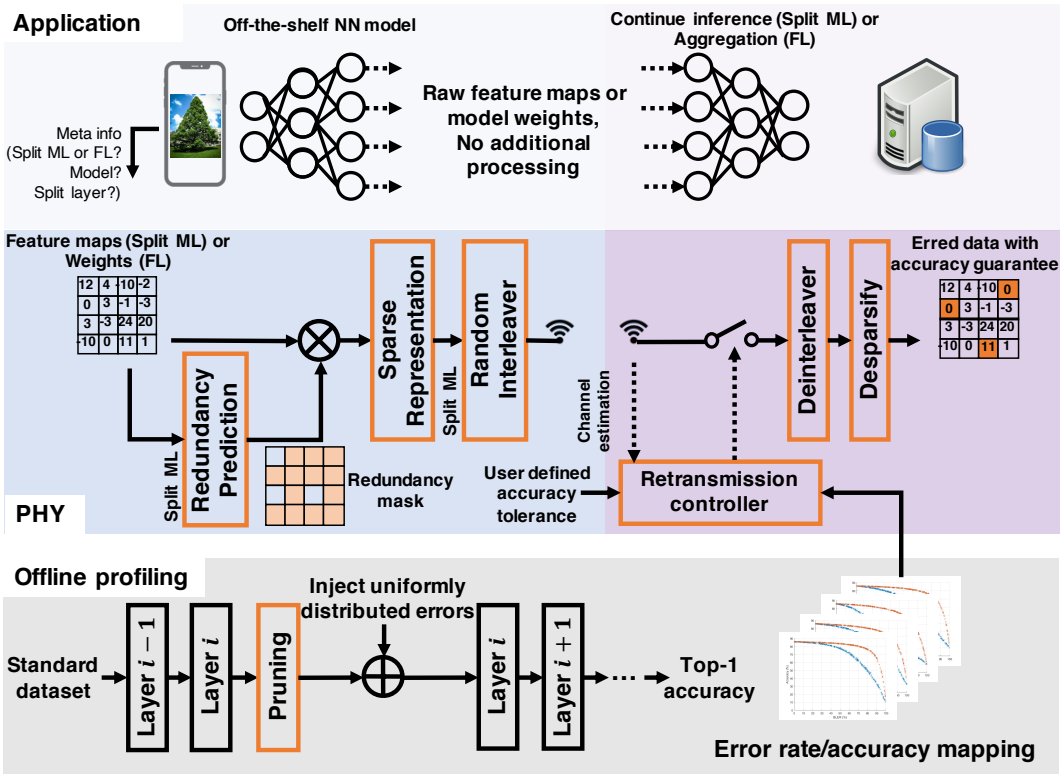


Figure 3.2: NeuroMessenger system overview.

only needs a lightweight application-independent profiling consisting of only the inference process which usually takes less than 1/1000 of time to run compared to the training.

3.3 System overview

The system architecture of NeuroMessenger is shown in Fig. 3.2. At runtime, the UE application runs off-the-shelf distributed models. It can pass to the low layer the raw ML data along with a small metadata indicating if the application is split ML or FL, what the NN model is, and the splitting point for split ML. NeuroMessenger, as part of the cellular-native stack, recognizes the metadata and acts on the ML data accordingly. For

split ML, NeuroMessenger first prunes the redundant data (Sec. 3.5.2), and then applies error tolerance coding (Sec. 3.5.1). As for FL, NeuroMessenger skips these two processes. The resulting data and metadata are then fed to the original cellular PHY layer and then transmitted to the basestation.

Upon receiving the data, the NeuroMessenger retransmission controller on the basestation estimates how many retransmission attempts are needed to balance the model accuracy and communication overhead. Specifically, the retransmission controller first uses the information in the metadata to find the corresponding error rate to accuracy mapping function G, which is profiled offline (Sec. 3.5.2). It also retrieves the latest uplink SNR measurement and estimates the corresponding error rate r. Then, the controller compares estimated accuracy under the current error rate $G(r)$ and a user-defined minimal tolerable accuracy a_u and calculates the maximum number of retransmission attempts n as follow:

$$n = \begin{cases} 0 & \text{if } G(r) > a_u \\ F(r, a_u) & \text{if } G(r) < a_u \end{cases} \tag{3.1}$$

where $F(r_1, r_2)$ calculates the minimal number of retransmission attempts to reduce error rate from r_1 to r_2. For example, $F(0.8, 0.5) = 2$ means that to reduce error rate from 0.8 to 0.5, the maximal number of retransmission attempts needed is 2. F is determined by the PHY layer MCS level and the channel condition, and can be derived by existing predictive models [84]. The basestation then signals the UE for retransmission until n attempts or the data passes parity check. By eliminating the retransmissions when $G(r) > a_u$ and reducing the number of retransmission attempts when $G(r) < a_u$, NeuroMessenger alleviates the communication overhead.

Notably, NeuroMessenger separates the ML data transfer from the main ML design, and thus it reduces the complexity of the distributed ML system design while still providing model-specific efficiency improvement. The lightweight offline profiling also enables easy and fast adaptations of rapidly evolving ML models to NeuroMessenger without any runtime overhead.

3.4 Error-tolerance in Distributed ML

In this section, we first characterize the error tolerance capability in a typical split ML setting. Then, based on the characteristics, we propose and verify two techniques to encode the intermediate data transfer and enhance error tolerance in generalized split ML models.

3.4.1 A dissection of neural network models

To study the error tolerance in split ML, we must first understand the structures of split ML and the neural network (NN) models. A NN model consists of multiple consecutive layers. During an inference task, a layer takes the output data from its previous layer, commonly referred to as *feature maps*, applies certain operations and feeds its own output feature maps to the next layer. Despite numerous variants, most commonly used convolutional neural network (CNN) models share the same 4-layer building blocks, as shown in Fig. 3.3.

Convolution (conv) layer convolves the input data or feature maps with a set

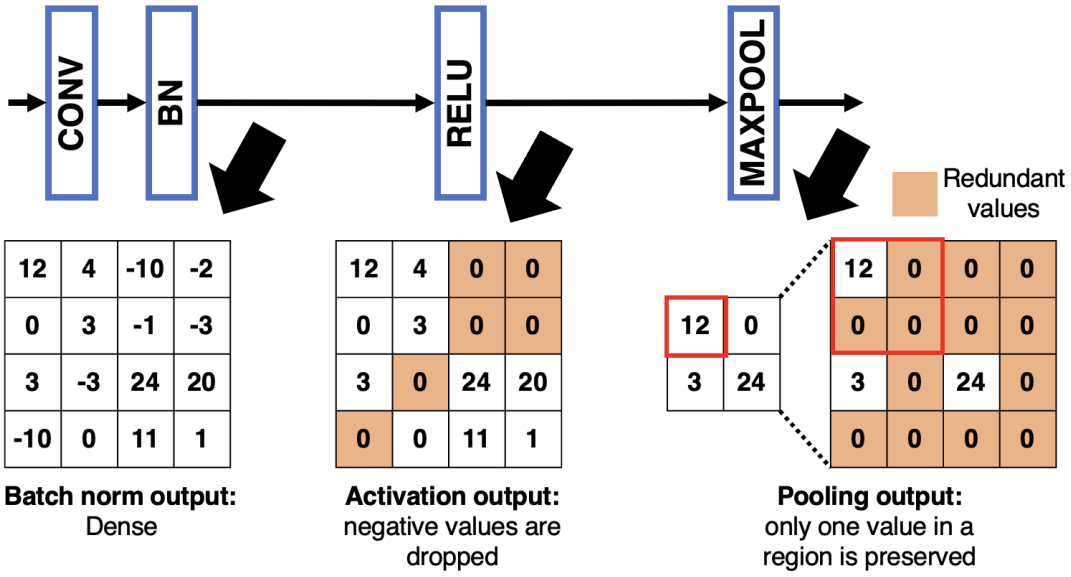


Figure 3.3: Layer composition of a typical neural network and a demonstration of feature map redundancy: most of parameters in the feature map from batch norm layer are dropped after the pooling layer.

of learned filters.

Batch norm (bn) layer normalizes a batch of feature maps.

Pooling (maxpool, avgpool) layer applies the maximum or average function over a region in the feature maps and reduces the region of parameters to a scalar.

Activation (sig, ReLu) layer applies a non-linear activation function to individual parameters and maps the negative or small valued parameters to zero.

In addition to the 4 basic layers in CNN, **fully connected layers**, which linearly combine all feature maps, are often used as a final classifier or an independent NN model for classification tasks.

3.4.2 Characterizing error tolerance in split ML

We first demonstrate the impact of errors of intermediate data transfer in split ML, with a pre-trained ResNet18 model [85]. ResNet18 is a popular image classification CNN model consisting of 54 layers. Due to its broad application and typical layer structure, it is often used as the benchmark for distributed ML designs [16, 55]. We assume an example split ML setting where a UE and a MEC split the model at the 5th layer over a noisy 5G link. The parameters in the feature maps are stored in standard 32-byte float type. To investigate the impact of different error patterns, we use two types of errors: (1) *burst errors* represents the common error pattern in packetized communications where an error corrupts an entire packet. To match our 5G link assumption, we set the length of a packet to 3777-byte blocks, a typical transport block size in 5G NR [86]. (2) *random errors* represents a more general but rarer error pattern where an error corrupts an random individual parameter in the feature maps. The corrupted parameters are treated as zeros in our characterization experiments. We vary the error rate and test the corresponding top-1 inference accuracy on the CIFAR100 dataset. Fig. 3.4 showcases the error rate to accuracy mapping when splitting at two example layers (layer 4 and 26). By examining the error rate to accuracy mapping at each layer, we have the following key observations:

The vanilla split ML has limited error tolerant capability. As shown in Fig. 3.4, the inference accuracy gradually decreases to 0 as the error rate increases. This implies the feature maps inherently possess error tolerance and may still produce correct inference result under a non-zero error rate. We have the same observation for the error

rate to accuracy mappings of all layers except for the fully connected layers. This inherent error tolerant capability is mainly caused by convolution and pooling layers after the splitting point which operate on local regions and often have "soft outputs", *i.e.* float point type. Even if a parameter is corrupted, the convolution and pooling operations in the layers after the splitting point can still produce similar values from other parameters in the same region and eventually dilute the impact of the corrupted parameter. Note that in both figures of Fig. 3.4, *the accuracy under burst errors is approximately linear with respect to the error rate*, with 7% accuracy loss for every 10% error rate increase. For a slightly noisy 5G uplink with 10% error rate [68], it will lead to 7% accuracy loss, which is usually the accuracy gap between an advanced NN model and a simple classifier. This implies that *the inherent error tolerance in ML data alone can hardly combat the block errors in practical communication systems without a major impact on inference accuracy.*

Error tolerance is layer-dependent. The error rate to accuracy mapping curve differs across layers, indicating that under the same link condition, splitting at different layers yields different accuracy. For example, at block error rate of 0.8, layer 26 and 4 show accuracy of 78% and 52%, respectively. Such gaps are observed for all other layers with accuracy differences ranging from $< 1\%$ to 50%. This seemingly obvious observation reveals the necessity for a layer-wise error rate to accuracy profiling in the cases where the splitting layer is dynamically selected.

Split ML is prone to burst errors. In Fig. 3.4, we see with the same error rate, the block errors cause far more accuracy loss than the random errors, *i.e.*, up to 20% for layer 4 and 50% for layer 26. For other layers in ResNet18, we also observe similar accuracy

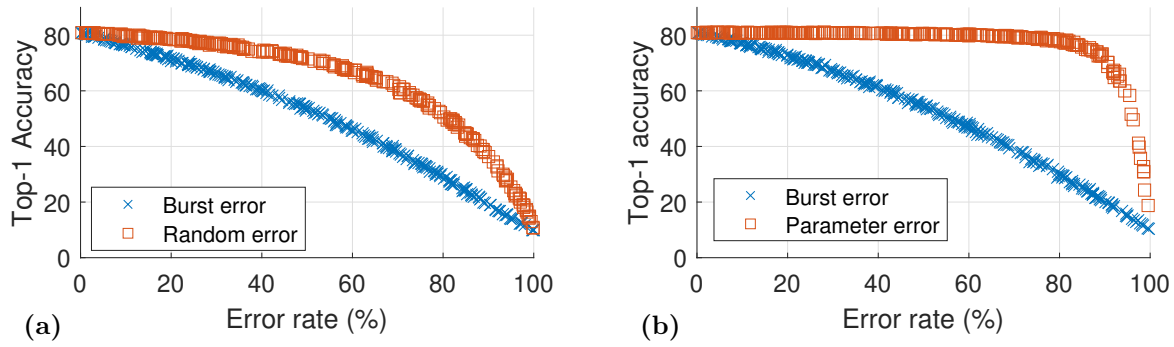


Figure 3.4: Top-1 accuracy with different error rate applied to the feature map from (a) layer 4, (b) layer 26.

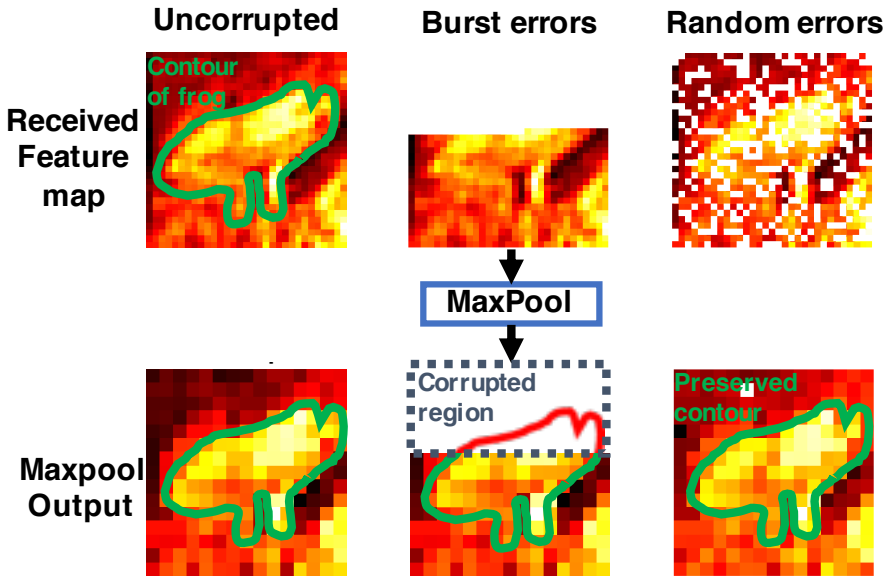


Figure 3.5: A demonstration of impact of different types of errors on feature maps: the top region of the frog shape is almost entirely corrupted by block errors, while the random error preserves the shape.

gaps ranging from 20% to 60%. Similar to the first observation, such vulnerability to burst errors originates from the convolution and pooling operations. Since these operations are applied to local regions, the output feature maps usually show a similar pattern to the input feature maps in spite of their different sizes. Fig. 3.5 shows a visualization of input and output feature maps at the 5th layer (maxpooling) of ResNet18 with an image of a frog as input. We see that the basic shape and the contour of the frog are preserved after the maxpooling layer, even though the feature map size reduces from 24×24 to 12×12. As a result, the *random errors in the input feature maps are usually diluted in the following layers*.

Burst errors, on the other hand, corrupt long sequences of parameters that usually span multiple local regions or even an entire input feature map. Since convolution and pooling layers can only produce zeros if all parameters in a local region are corrupted (treated as zero in our experimental characterization), the impact of burst errors is often preserved on the following layers. Fig. 3.5 showcases such phenomenon. We see that under burst errors, the top half of the frog contour is corrupted even after a maxpooling layer. In comparison, the random errors only corrupted a small region and the frog contour is preserved. Hence, the burst errors are more likely to cause a large accuracy loss than random errors.

3.4.3 Characterizing error tolerance in FL

As mentioned in Sec. 3.1, instead of feature maps, the clients in FL transfer the model weights. Hence, it is reasonable to expect that FL possesses different error tolerance

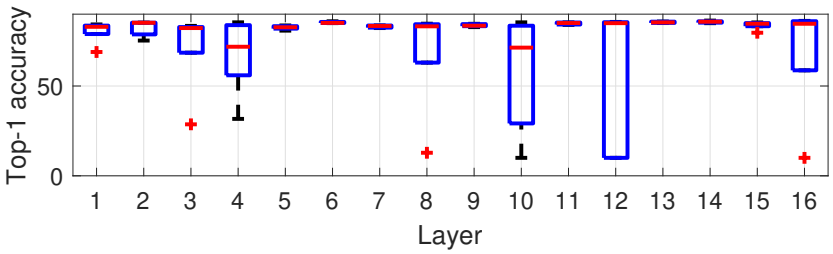


Figure 3.6: Per-layer accuracy of VGG11 FL under BLER=0.1. We see the ranges of accuracy of batch norm layers (3, 4, 8, 10, 12, 16) exceed 20 percent, indicating that the batch norm layers are highly sensitive to errors.

characteristics than split ML. We investigate the impact of errors on a FL system with the VGG11 model. We assume a typical FL system with 20 participating devices, each training with $\frac{1}{20}$ CIFAR-100 dataset that follows i.i.d. distribution. The model weights are aggregated using the widely-adopted FedAvg algorithm [87] in a synchronized manner, *i.e.*, in each epoch, one client only uploads model weights one time. To avoid the impact of heterogeneous links, we assume the FL system selects clients with similar link conditions and all clients share the same error rate. To investigate layer-wise error tolerance, we only apply errors to one layer's weights for each trial. Note that the activation and pooling layers are not considered in the experiment since they do not have weights. The model is trained over 50 epochs and then tested with the CIFAR-100 testing set. For each layer and error rate, we repeat the experiment 10 times. From the results, we make the following key observations:

Batch norm layers in FL are not error-tolerant. Fig. 3.6 shows the top-1 accuracy when applying random errors with 10% error rate to each layer. Although most layers maintain a stable accuracy for all 10 trials, layer 4, 8, 10, and 12 show a wide range

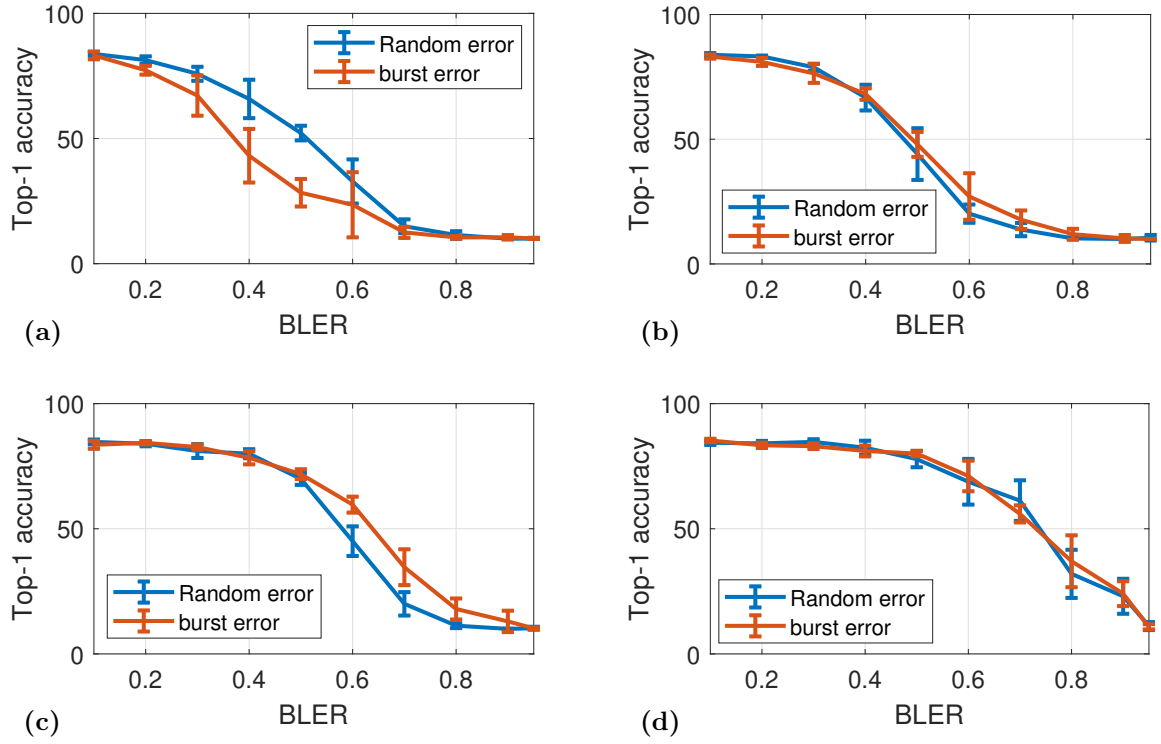


Figure 3.7: Top-1 accuracy of VGG11 trained in FL under different block error rate applied to (a) 8st, (b) 11th, (c) 18th, and (d) 22th layers (conv).

of accuracy fluctuations up to 70%. Upon further inspection, we found that all layers with larger than 20% accuracy fluctuations are batch norm layers. Fig. 3.8 further shows the accuracy when the two batch norm layers experience different error rates. We see such fluctuations exist persistently. Unlike the convolution layers' weights which only affect one small region of a feature map, the weights in the batch norm layers are applied to a batch of feature maps. Consequently, the weight errors have a larger impact on accuracy. This result indicates that *ideally batch norm layers' weights should be transferred error-free.*

FL is prone to both burst errors and random errors. To investigate the impact of error patterns, we compare the accuracy from two different types of error patterns.

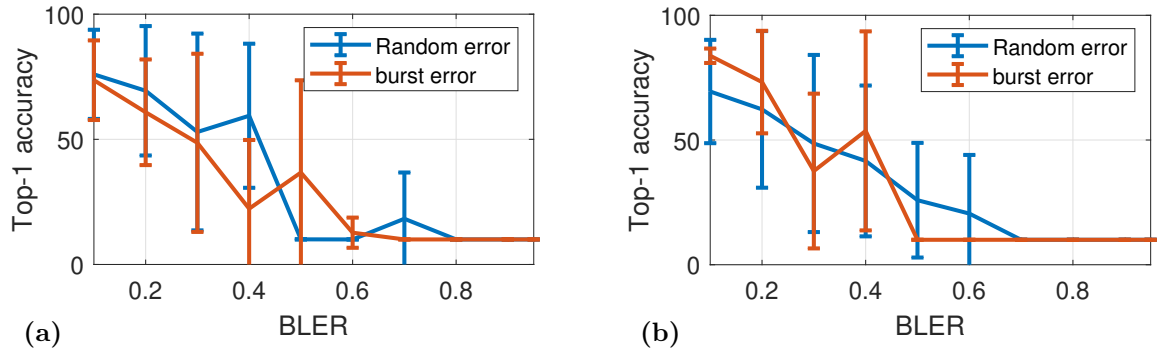


Figure 3.8: Top-1 accuracy of VGG11 trained in FL under different block error rate applied to (a) 16st, (b) 19th layers (batch norm).

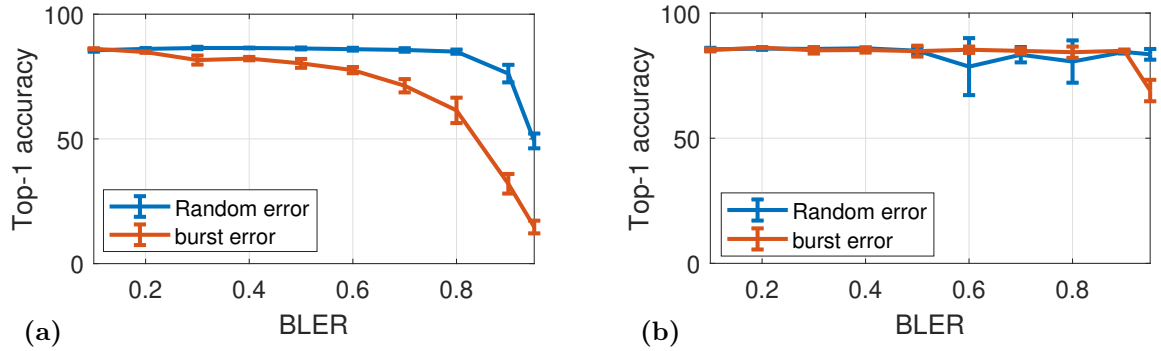


Figure 3.9: Top-1 accuracy of VGG11 trained in FL under different block error rate applied to (a) 25th, (b) 26th layers (fully connected).

As shown in Fig. 3.7 and 3.9, we see that for both convolution and fully connected layers, random error and burst error have similar impacts on model accuracy. This is because the training process in ML has a forward phase and a backward phase. The forward phase, essentially the same as the inference phase, computes the inference result and the corresponding loss from the first layer to the last layer, while the backward phase uses the loss to compute the gradient of each layer propagating from the last layer to the first layer. Similar to split ML, the convolution and pooling layers dilute the errors in the forward phase and hence reduces its negative impact on the model accuracy. However, the same layers spread out errors to more parameters during the backward phase which amplifies the impact of errors on the model accuracy. As a result, there is no discernible advantage of random errors over burst errors. This means *FL's error tolerance cannot be improved by interleaved coding.*

3.5 NeuroMessenger Operations

3.5.1 Error tolerance Enhancing Coding

Interleaved Coding The foregoing experiments hint that split ML suffers more accuracy loss from burst errors than random errors under the same error rate. In practice, however, due to the packetization operation and wireless channel coherence, the majority of the errors are in the form of burst errors [88]. To alleviate the impact of such burst errors, we adopt an interleaved coding method to encode the feature maps. Interleaved coding is a family of codes aiming to convert the burst errors to random errors by interleaving the

data sequence [89]. The basic idea is that the interleaving operation redistributes a long sequence of errors across many short separated bursts.

To showcase the interleaved coding, we apply it to the ML data in the burst error experiment in Sec. 3.4B and compare the top-1 accuracy. Without loss of generality, we use random interleaved coding [90]. The code generates a randomized permutation whose size equals the number of float point numbers in ML data and reorders the numbers in the data accordingly. To understand the result across different models, we add VGG11 [91], another popular image classification NN, to the experiment. We choose 4 different split points for each model, each at a quadrate point of the model. Fig. 3.10 and 3.11 show the the results.

We see that interleaved coding improves accuracy by up to 190% for all split points in ResNet18 and first 3 points in VGG11 (we will explain the last point later). Specifically, the last two points in ResNet18 show less than 1% accuracy loss when error rate is less than 50% while the baseline loses 45% accuracy at the same error rate. The result implies *the interleaving technique substantially improves the burst error tolerance for most layers in CNN models.*

Importance-based Coding Unlike the regional operations in the convolution and pooling layers, a fully connected layer operates on all parameters in feature maps and the output is a linear combination of all parameters whose weights are obtained during the training phase. Instead of the patterns and shapes in feature maps, the impact on accuracy from fully connected layers is determined by the absolute value of individual parameters. Recall that interleaved coding only redistributes the long sequence of errors

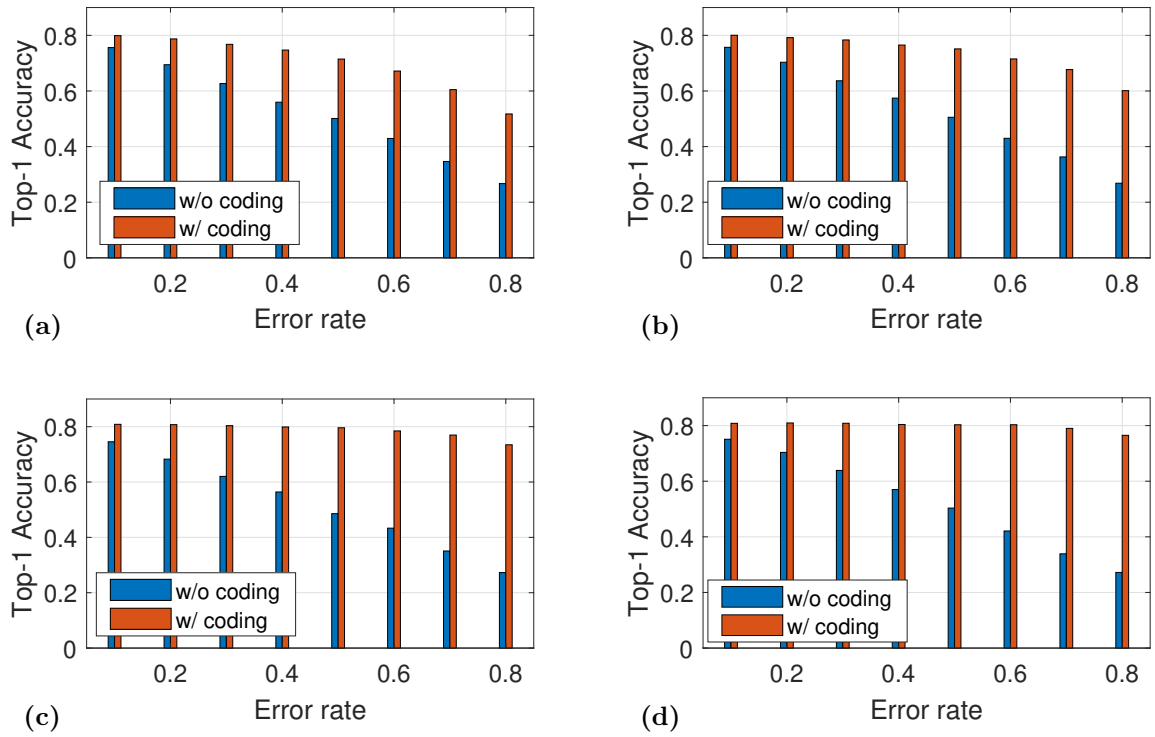


Figure 3.10: Top-1 accuracy of ResNet18 under different block error rate with the split point after (a) first, (b) second, (c) third, and (d) forth residual module.

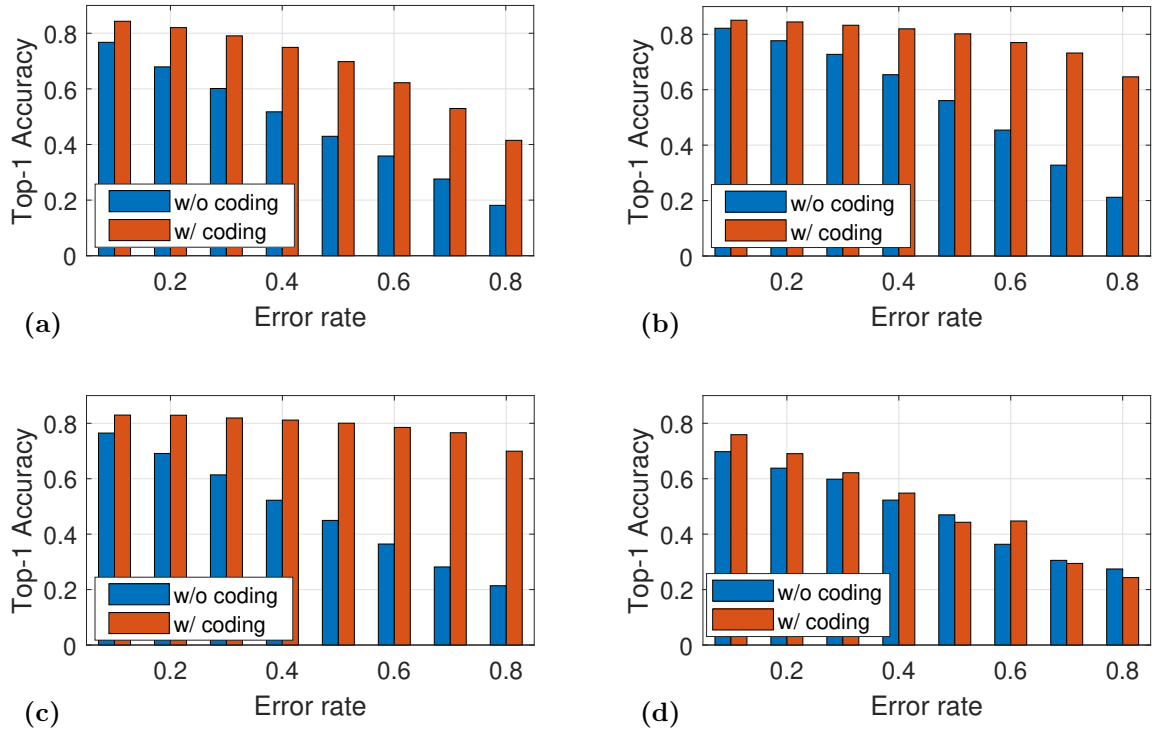


Figure 3.11: Top-1 accuracy of VGG11 under different BLER with the split point after (a) 1st, (b) 5th, (c) 19th, and (d) 26th layer.

without reducing the number of errors. As a result, the *interleaved coding does not improve the error tolerance for fully connected layer*. This explains the result in Fig. 3.11(d) where the interleaved coding shows similar accuracy as the baseline.

To enhance error tolerance for fully connected layers, we adopt importance-based coding, a family of codes that provide Unequal Error Protection (UEP) capability for data with different importance. The basic idea is that a corrupted high-valued parameter will have more impact on accuracy than a corrupted low-valued parameter (*e.g.*, near 0). Hence the high valued parameters are more important and should be protected against high error rates by UEP.

To showcase the importance-based coding, we repeat the previous burst error experiment for the DeepSpeech2 model [92]. DeepSpeech2 is a recurrent neural network (RNN) speech recognition model consisting of fully connected and activation layers. Similar to ResNet18, it is often used as a benchmark to evaluate distributed ML systems [93]. To show the maximum possible improvement, we assume an ideal importance-based coding scheme that ensures the error rate of a particular parameter is inversely proportional to its value while the total error rate is constant. In practice, the importance coding applies UEP techniques such as Hadamard matrix [94], network slicing [95], and repetition [96], to the high-valued parameters to reduce their error rate under the same overall error rate. Fig. 3.12 shows the Word Error Rate (WER) performance for 4 splitting points at each quadrate point. A higher WER means lower speech recognition accuracy We see the ideal importance-based coding generally reduces WER by up to 60%. When block error rate (BLER) is less than 50%, the WER with ideal importance-based coding is maintained at

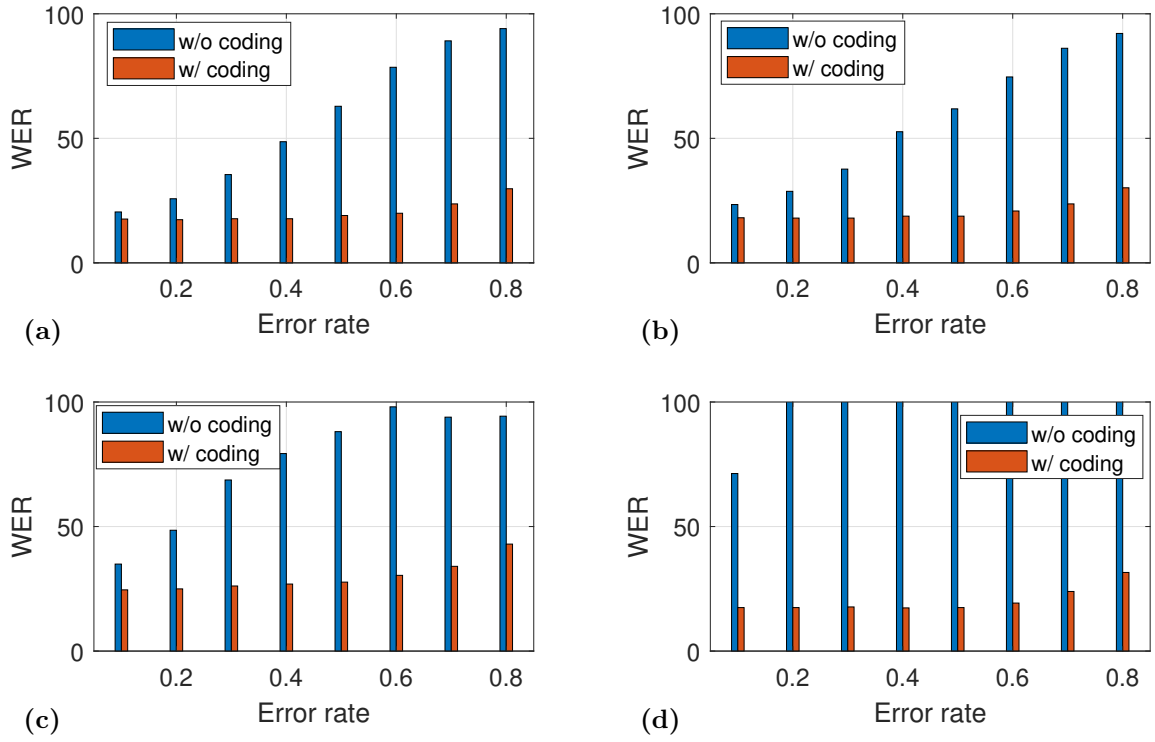


Figure 3.12: Word error rate of DeepSpeech2 under different block error rate with the split point after (a) first, (b) second, (c) third, and (d) forth splitting point.

$< 20\%$, less than 1% increase from the 0% block error rate case. The result shows that the ideal *importance-based coding effectively enhances the error tolerance for fully connected layers.*

3.5.2 Additional Operations

Layer-wise error-tolerance profiling As demonstrated in Sec. 3.4 and 3.4.3, the accuracy performance of a distributed ML model depends on the link error rate, as well as on which layer the error occurs. To ensure a certain level of accuracy, we need to profile the mapping between the error rate and inference accuracy, and use the profile to

estimate accuracy under a given link condition at runtime. We first reuse the experimental settings in Sec. 3.4 and 3.4.3 where the ML data experience random errors. For split ML models, since the splitting point may change at runtime, we profile the error rate to accuracy mapping for each layer offline. For FL models, we only profile a single error rate to accuracy mapping by applying the errors to the weights of all except the batch norm layers. We vary the error rate and record the corresponding top-1 testing accuracy. Suppose the error rates are $\{r_1, r_2, ..., r_L\}$, with corresponding accuracy $\{a_1, a_2, ..., a_L\}$. To approximate the error rate to accuracy mapping function, we empirically choose an exponential function G to fit the mapping, so that the L1 distance between the inferred accuracy and measured accuracy is minimized.

$$G = \min_{G} \sum_{i=1}^{L} |a_i - G(r_i)| \tag{3.2}$$

Since the error tolerance strongly depends on error rate (Sec. 3.4), we can thus accurately estimate the accuracy with G, for a given error rate (derived from the link SNR). The profiling is performed offline and only needs to be done once for a given ML model. The profiling latency is determined by the device's computation power as well as the size and depth of the ML model. We observe a profiling latency of $< 10s$ for split ML and $< 10min$ for FL with 54-layer ResNet18 model on a server with Nvidia RTX2080 TI GPU.

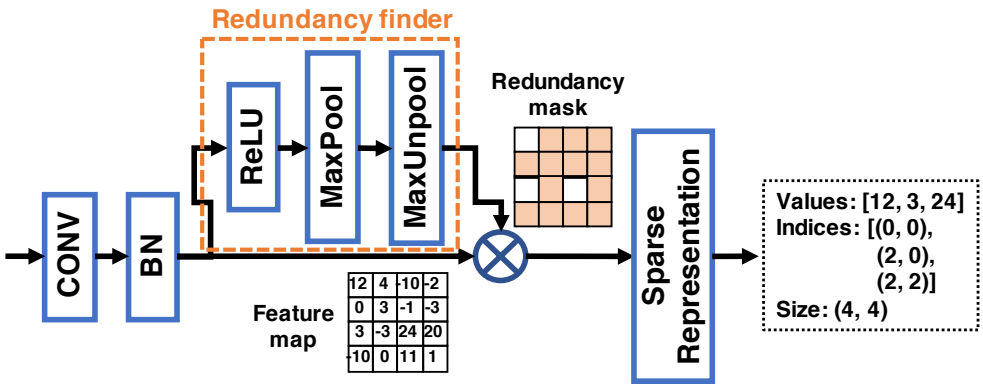


Figure 3.13: An illustration of feature map pruning: (1) generate redundancy mask, (2) Multiply redundancy mask to the feature maps, (3) Convert pruned feature maps to sparse representation.

Table 3.1: The top-1 inference accuracy and the end-to-end latency performance of NeuroMessenger FL and baselines under a Matlab simulated noisy 3GPP NR uplink.

	VGG11 - CIFAR100			ResNet18 - CIFAR100			DeepSpeech2 - LibriSpeech		
	t_{comm}	Acc.	retr. ?	t_{comm}	Acc.	retr. ?	t_{comm}	WER	retr. ?
Ours	8.80s (-35.6%)	84.0% (-2.3%)	None	0.77s (-35.8%)	82.6% (-4.0%)	None	20.12s (-38.8%)	18.7% (-6.3%)	None
HARQ	13.66s (-0.0%)	86.0% (-0.0%)	Yes	1.20s (-0.0%)	80.0% (-0.0%)	Yes	31.36s (-0.0%)	17.6% (-0.0%)	Yes

Table 3.2: The top-1 inference accuracy and the end-to-end latency performance of NeuroMessenger split ML and baselines under a Matlab simulated noisy 3GPP NR uplink.

	Split point 1			Split point 2			Split point 3			Split point 4		
	t_{comm}	Acc.	retr. ?	t_{comm}	Acc.	retr. ?	t_{comm}	Acc.	retr. ?	t_{comm}	Acc.	retr. ?
VGG11 - CIFAR100												
Ours	36.0ms (-91.9%)	79.7% (-8.1%)	None	26.7ms (-87.7%)	83.3% (-3.1%)	None	4.8ms (-91.2%)	82.0% (-11.1%)	None	<1ms (-93.0%)	65.8% (-23.5%)	None
Raw	433.8ms (-0.0%)	68.2% (-20.7%)	None	216.8ms (-0.0%)	85.2% (-0.9%)	None	54.3ms (-0.0%)	83.0% (-3.5%)	None	13.6ms (-0.0%)	65.4% (-24.0%)	None
Pruned	33.8ms (-92.2%)	60.14% (-30.1%)	None	24.5ms (-88.7%)	72.8% (-15.3%)	None	2.6ms (-95.2%)	61.4% (-28.6%)	None	< 1ms (-93.0%)	59.8% (-30.5%)	None
Pruned-HARQ	46.6ms (-89.3%)	86.0% (-0.0%)	Yes	38.0ms (-82.4%)	86.0% (-0.0%)	Yes	4.1ms (-92.4%)	86.0% (-0.0%)	Yes	< 1ms (-93.0%)	86.0% (-0.0%)	Yes
ResNet18 - CIFAR100												
Ours	16.0ms (-39.6%)	76.8% (-4.0%)	None	6.0ms (-55.9%)	78.4% (-2.0%)	None	2.5ms (-63.2%)	80.0% (-0.0%)	None	2.1ms (-36.4%)	80.0% (-0.0%)	None
Raw	26.5ms (-0.0%)	66.5% (-16.9%)	None	13.6ms (-0.0%)	78.6% (-1.8%)	None	6.8ms (-0.0%)	68.3% (-14.6%)	None	3.3ms (-0.0%)	68.3% (-14.6%)	None
Pruned	14.9ms (-43.8%)	76.8% (-4.0%)	None	3.8ms (-72.1%)	63.7% (-20.4%)	None	2.2ms (-67.6%)	62.0% (-22.5%)	None	1.7ms (-48.5%)	63.9% (-20.1%)	None
Pruned-HARQ	23.1ms (-12.8%)	80.0% (-0.0%)	Yes	6.0ms (-55.9%)	80.0% (-0.0%)	Yes	3.3ms (-51.5%)	80.0% (-0.0%)	Yes	2.6ms (-21.2%)	80.0% (-0.0%)	Yes
DeepSpeech2 - LibriSpeech (Accuracy represented by WER)												
Ours	3.5ms (-62.8%)	17.7% (+0.5%)	None	3.5ms (-62.8%)	18.0% (+2.2%)	None	3.5ms (-62.8%)	25.0% (+42.0%)	None	3.5ms (-62.8%)	17.7% (+0.5%)	None
Raw	9.4ms (-0.0%)	22.3% (+26.7%)	None	9.4ms (-0.0%)	27.6% (+56.8%)	None	9.4ms (-0.0%)	47.2% (+168.2%)	None	9.4ms (-0.0%)	80.1% (+355.1%)	None
Pruned	3.0ms (-62.8%)	35.4% (+101.1%)	None	3.0ms (-62.8%)	37.6% (+113.6%)	None	3.0ms (-62.8%)	68.7% (+290.3%)	None	3.0ms (-62.8%)	100.0% (+468.2%)	None
Pruned-HARQ	4.6ms (-51.1%)	17.6% (-0.0%)	Yes	4.6ms (-51.1%)	17.6% (-0.0%)	Yes	4.6ms (-51.1%)	17.6% (-0.0%)	Yes	4.6ms (-51.1%)	17.6% (-0.0%)	Yes

Feature map pruning The aforementioned error-tolerance techniques effectively reduce the need for frequent retransmissions, but do not reduce the data size which may induce a large initial transmission overhead. To further improve efficiency, we introduce a feature map pruning design for split ML. It is observed that most parameters (*i.e.* the float point numbers) in the feature maps are dropped by the first maxpooling and activation layers after the split point, *i.e.*, their values do not affect the inference result of the ML model [17]. Hence, these redundant parameters can be safely removed to reduce the ML data size. However, it is challenging to determine which parameters are redundant since it depends on the model input and cannot be predicted without running the layers first.

To efficiently find the redundant parameters, we introduce a *quantized NN based redundancy prediction* scheme. The general idea is that since the parameters are made redundant by the first activation and pooling layers after the split point, we can feed the feature maps to such layers to find which parameters are redundant. It is time-consuming to run full layers so instead, we use their quantized versions which have the same operations. As shown in Fig. 3.13, when a feature map is generated at the split point, it is first quantized and then fed to a redundancy finder branch, which consists of the first activation and pooling layers after the split point, and an unpooling layer [97] to map the pooling layer's output to their corresponding positions in the original feature maps. The output of the redundancy finder branch is a feature map that has the same size as the original feature map, but with redundant parameters converted to zero. We replace all non-zero parameters in this feature map output to 1 and multiply it element-wisely with the original feature maps. The feature map becomes sparse after such a pruning step and

can be efficiently stored using standard sparse representation [98] with a data size of only a fraction of the original feature maps.

3.6 Evaluation

3.6.1 Experimental setup

We evaluate NeuroMessenger on a Matlab-based 5G link-level simulation framework [99], which simulates an end-to-end 5G NR uplink with complete MCS implementations as well as 3GPP compliant channel models. By default, we use 16QAM modulation with 490/1024 LDPC code rate. Varying the link SNR leads to different levels of error rate. The OFDMA uses 20MHz bandwidth with 30KHz subcarrier spacing. We adopt a 2 $\times$ 2 MIMO setting with 2 PUSCH layers. The transport block size is set to 30216 bits. For the physical channel, we use 3GPP CDL-C clustered delay line channel [100] which represents a generic multi-path channel. Specifically, we set the SNR of the channel to -2dB by default to simulate a noisy link. Using a high order MCS, even with a relatively high SNR, may result in a similarly noisy link.

Models and dataset. We evaluate NeuroMessenger on two of the most widely deployed applications: image classification and speech recognition. For image classification, we choose ResNet18 and VGG11 to represent state-of-art CNN models. For testing purposes, we use CIFAR100, a standard image classification data set with 100 classes and 100 test images for each class. For speech recognition, we use the DeepSpeech2 model applied on the LibriSpeech dataset, a large corpus of reading English speech containing 5

hours of testing speech.

Baselines For split ML experiments, we compare the performance of NeuroMessenger with the following three baselines: *(i) Raw* does not perform any additional processing on the ML data. *(ii) Pruned* prunes the redundancy of the ML data but does not perform the two coding schemes that enhance error tolerance. *(iii) Pruned-HARQ* prunes the redundancy and leverages the built-in HARQ retransmission mechanism in 5G. The retransmission version (RV) in HARQ, *i.e.*, the maximal number of retransmission attempts, is set to 16. For FL experiments, since feature map pruning and error-tolerant enhancement schemes are not applicable, we use only one baseline: *HARQ*, *i.e.*, the default retransmission scheme in 5G instead of the retransmission controller in NeuroMessenger.

3.6.2 End-to-end performance

Table **??** and 3.1 show the inference accuracy and end-to-end latency performance for split ML and FL. In this experiment, we set the maximal tolerable accuracy loss of NeuroMessenger to 80% (−6% compared to the original model) for image classification models, and the maximal WER increase to 20% (+3.4% comparing to the original model) for the speech recognition model. Compared to the raw baseline, the pruned baseline reduces up to 95.2% transmission times accompanied by 9.3% to 113.1% higher loss in accuracy. This means *NeuroMessenger's redundancy pruning design effectively reduces the intermediate data size*. But it sacrifices the error tolerance capability, implying the need for error enhancing coding. The pruned-HARQ baseline achieves the best accuracy due to frequent retransmissions which also causes 20% more communication latency than

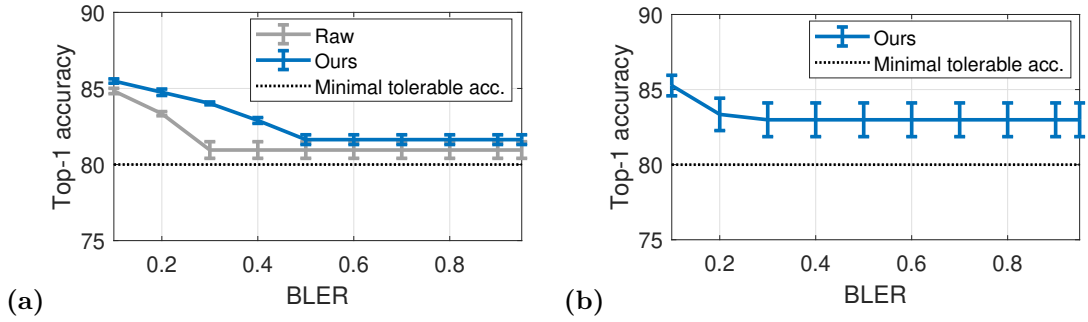


Figure 3.14: Top-1 accuracy of (a) split ML (VGG11 split at 15-th layer) and (b) FL (VGG11), as the block error rate of the link increases.

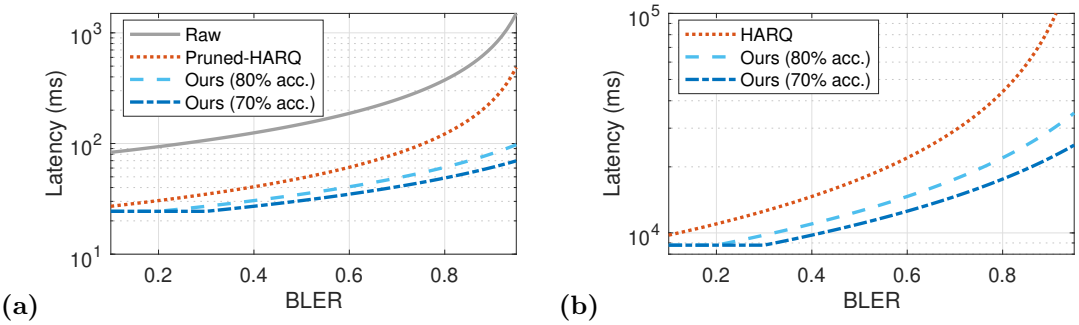


Figure 3.15: End-to-end running latency of (a) split ML (ResNet18 split at 15-th layer) and (b) FL (ResNet18), as the block error rate of the link increases

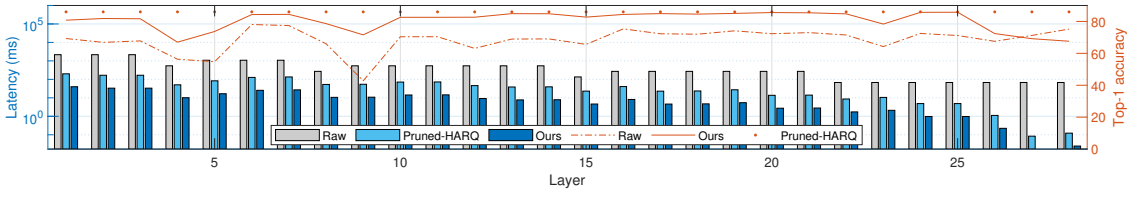


Figure 3.16: End-to-end running latency and top-1 accuracy at BLER=0.2 when splitting at each layer in VGG11.

the pruned baseline. *Our NeuroMessenger design strikes the best balance between the accuracy and latency*: its transmission time only increases 1%-5% on top of the pruned baseline due to the computation time of coding while the accuracy is maintained above the maximal tolerable accuracy most of time, which is 0.9%-30.5% higher than the raw and pruned baselines. In summary, the experiment shows the performance advantages of NeuroMessenger over the baselines in split ML and the necessity of the coding mechanisms for enhancing error-tolerance.

3.6.3 Impact of Link Conditions

To investigate the performance of NeuroMessenger under different link conditions, we vary the SNR of the 5G link so that the block error rate increases from 0.1 to 0.95. Fig. 3.14 and Fig. 3.15 show the top-1 accuracy and latency results of ResNet18 in split ML and FL, respectively. We set the minimum tolerable accuracy to 80% for both split ML and FL. In Fig. 3.14, we see accuracy slightly decreases to 82% as the BLER increases. It remains at the same level as the NeuroMessenger retransmission controller determines that error tolerance can no longer satisfy the accuracy requirement and enables retransmission. Note that even for the raw baseline in Fig. 3.14, the NeuroMessenger retransmission controller can still guarantee a minimal latency of 80% by enabling retransmission earlier.

In Fig. 3.15(a), we see the latency of raw and pruned-HARQ baselines increases exponentially with BLER as the number of retransmission increases. The latency of NeuroMessenger stays constant until BLER= 0.5 due to the absence of retransmissions. Note that if we lower the minimal tolerable accuracy to 70%, the latency stays constant until

BLER= 0.9. The FL's latency in Fig. 3.15(b) shows a similar trend but with a lower constant latency.

In summary, this experiment shows that *the NeuroMessenger retransmission controller can effectively guarantee the accuracy under a wide range of block error rate, and NeuroMessenger keeps the communication latency constant when the block error rate is within the error tolerance of the ML data*, which is determined by the ML model, layer, and minimal tolerable accuracy.

3.6.4 Impact of Split Point

Recall that in split ML, the error tolerance is different across layers (Sec. 3.4). Hence the effectiveness of NeuroMessenger may vary with different choices of split point. To investigate this effect, we reuse the experimental setup in Sec. 3.6 and split a VGG11 model at each layer. Fig. 3.16 shows the latency and corresponding accuracy performance. We see that NeuroMessenger achieves 40%-99% latency reduction comparing to the raw baseline and 20% comparing to the pruned-HARQ baseline for all layers. Although pruned-HARQ achieves the highest accuracy, NeuroMessenger achieves similar accuracy for most layers and significantly higher accuracy than the raw baseline. In summary, this experiment and the experiment in Sec. 3.5.1 jointly show that the advantage of NeuroMessenger applies to a wide range of split points on typical NN models.

3.7 Conclusion

In this paper, we have explored the error tolerant capability in distributed ML data transfer and its implications on communication efficiency. We characterize the error tolerance of various popular distributed ML systems and develop a novel system, NeuroMessenger that enhances and utilizes such error tolerance. We believe our work envisions a new direction towards efficient distributed ML over wireless edge networks.

Chapter 3 contains material from "NeuroMessenger: Towards Error Tolerant Distributed Machine Learning Over Edge Networks" by Song Wang and Xinyu Zhang, which appears in the IEEE International Conference on Computer Communications, 2022. The dissertation author was the primary investigator and author of this paper.

Chapter 4

Omnidirectional Millimeter-Wave Coverage for ML Model Splitting

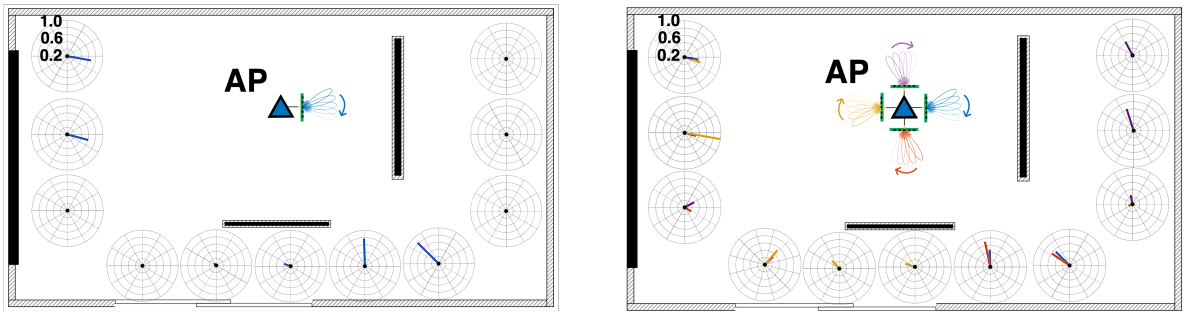
4.1 Introduction

The emerging wireless infrastructure is facing a massive mobile traffic demand [101, 102, 103, 104], driven by billions of upcoming Internet of Things and immersive multimedia applications [105, 106, 107, 108, 109]. Due to the spectrum crunch in legacy low-frequency bands, both the wireless local area and cellular network standards have been incorporating millimeter-wave (mmWave) technologies (*e.g.*, 802.11ad and 5G NR) to meet the looming challenge of mobile traffic overload. Ideally, one would expect the mmWave technologies to provide WiFi or LTE-like seamless coverage. But mmWave signals have orders of magnitude higher attenuation loss, which has to be compensated through high-gain phased array antennas. The directional gain of a phased array is,

empirically, proportional to the number of antenna elements [110, 111, 112, 113]. Hence, to provide sufficient mmWave coverage, an intuitive way is to simply increase the phased array size.

However, the high directionality brings two new challenges to mmWave networks: *(i) Beam management overhead.* A phased array may generate hundreds of beam patterns with main lobes pointing to different directions. Ideally, by rapidly scanning through the beams, it can approximate the behavior of an omni-directional antenna. Yet, when the receiver is mobile or when the line-of-sight (LoS) is blocked by obstacles, such a trial-and-error scanning may incur huge overhead in finding an alternative beam [114, 115, 116]. Recent years witnessed substantial research in designing efficient algorithms to identify the optimal beam directions under link dynamics [117, 118, 119, 120]. *(ii) Limited field-of-view (FoV) coverage.* A phased array has limited FoV (typically narrower than 120° [121, 122, 123]), due to the intrinsic properties of its patch antenna elements. This problem remains largely underexplored. Very recently, a Pia system [124] was proposed to combine multiple access points (APs) to expand the FoV [124], but this requires dense deployment, which is often not economically feasible, and involves tight coordination among distributed APs.

A more viable approach to overcoming the FoV limitation is to aggregate standard phased arrays to form an "array of phased arrays" (APA). In an APA mmWave radio, multiple phased arrays are co-located, sharing the same RF chain but facing different angles to jointly cover 360° in azimuth or elevation. The combined coverage may provide more *multipath diversity*, *i.e.*, signals may traverse different paths through reflections,



(a) (b)

Figure 4.1: Coverage and multipath diversity under the same power constraint: (a) Single array. (b) 4-array APA.

making it easier to save a link under blockage. The APA architecture has been adopted recently by both 802.11ad [125, 126] and 5G NR devices [127, 128, 129, 130, 131, 132]. Apart from coverage, APA also offers advantages in cost and efficiency. This is because beyond certain physical dimensions of the radio package, the feed network losses (between the radio RF front-end and a single giant phased array) would negate the benefits of having higher array gains [121]. In addition, all the phased arrays can share the same codebook, thus reducing the on-chip register/memory requirements, which accounts for non-trivial cost on a radio device[133]. Overall, APA makes it practical to scale to large phased arrays [134].

Unfortunately, APA also incurs new challenges to mmWave network design. *First*, due to the regulation constraint on emission power, not all phased-arrays can be turned on simultaneously. So a node has to decide on not only which beam to use for each array, but also how many and which arrays to activate. The decision space easily escalates to

an intractable scale. To our knowledge, no existing work attacked such a problem of array/beam management. *Second*, the APA should leverage its advantages in multipath diversity, to efficiently recover from link outage caused by mobility, blockage, or a mix of both. This would require it adapt the beam/array selection in real-time, with minimum protocol level overhead. *Third*, not all phased arrays' signals combine in a coherent way, so a straightforward way of turning on multiple arrays may even lead to lower link quality than a single array.

In this paper, we propose a novel system called *X-Array*, which explores the challenges and opportunities from APA, through three major design components. *(i)* We propose an optimization-driven array/beam selection algorithm to maximize the link quality under the power budget constraints. The solution is formulated as a look-up table, which is generated in an offline one-time manner and maps the dominant signal paths' angle-of-departure (AoD) to the optimal array/beam combination. At run-time, the APA node only needs to run a simple AoD estimation algorithm, leveraging the periodic beacon scanning defined in mmWave standards such as 802.11ad. *(ii)* We apply a low-overhead *dynamic co-phasing* algorithm to the different transmit arrays, so that their signals can coherently combine at the receiver, with very infrequent feedback. This method can maximize the combined directional gain (under regulation constraint), while maintaining as wide beamwidth as possible, to make the link more resilient under mobility. *(iii)* We design a link recovery mechanism that leverages the multi-array architecture to efficiently and accurately find alternative arrays/beams when the strong path disappears (*e.g.*, due to blockage) or reappears, with minimal overhead.

We have implemented X-Array on a commodity single RF chain 802.11ad AP supporting up to 8 arrays. X-Array runs at the device's user space, so it does not require any hardware modification. We have conducted experiments to verify X-Array in diverse radio environments, including indoor and outdoor, with different levels of mobility, multi-path conditions, and blockage dynamics. The results demonstrate that: *(i)* X-Array can approach the best array/beam combinations for all the settings, and the corresponding overhead is comparable with single-array solutions. *(ii)* X-Array can efficiently update the array/beam selection under link dynamics caused by mobility and blockage. *(iii)* X-Array can correctly apply the co-phasing factors to the multiple phased arrays to maximize the advantages of APA, while respecting the transmit power constraint. *(iv)* The joint coverage achieved through X-Array is comparable to an oracle solution that exhaustively searches across the decision space. By dynamically selecting and switching among 8 arrays or even 4 arrays, X-Array can approximately achieve omni-directional coverage in a sophisticated environment with random blockage. In contrast, a naive solution with a single array or two fixed arrays leave many blind spots with extremely low bit-rate.

Although commercial APA 802.11ad radios already exist [126, 125], they typically turn on all arrays which is far from optimal and may even be worse than a single-array (Sec. 4.5). X-Array represents the first system to fully exploit the advantages of APA. Our main contributions can be summarized as follows. *(i)* An efficient way to jointly manage multiple phased arrays and their beams to maximize link quality; *(ii)* A multi-array joint beamforming mechanism to ensure coherent combination of the multiple array's signals; *(iii)* A link recovery mechanism to ensure the robustness of APA under blockage; *(iv)*

Implementation and validation of X-Array on a commodity 802.11ad APA radio.

Our implementation of X-Array essentially converts a commodity multi-array 802.11ad radio [126] into a partially programmable experimental platform. Unlike recently developed mmWave software radios [135, 136], this platform can only run the 802.11ad MAC/PHY and does not provide channel state information (CSI). But it is less costly, and it allows for selecting beam, codebook, and arrays. Latest development of the platform will be documented in [137], and instructions for using the platform will be provided upon reasonable requests.

4.2 Motivation and Challenges

In this section, we conduct preliminary experiments to demonstrate the potential benefits of the APA mmWave radio, and the practical challenges in harvesting the benefits.

4.2.1 Potential Advantages of APA

We first investigate the unique channel characteristics of APA in comparison with single-array. Our experiments run on an off-the-shelf 802.11ad APA radio. The radio supports multiple arrays, each being a 6×6 uniform planar array with around 120° FoV. More detailed hardware specifications will be introduced in Sec. 4.4. As a benchmark experiment, we place a single-array AP and a 4-array AP in the middle of a 11m×6m indoor meeting space. For a fair comparison, we *configure the two APs to use the same set of beams on each array, and the same total power constraint.* Two poster boards (with

metal plate on the back) stand nearby, representing reflectors/obstacles just as in a typical environment.

During the experiment, a user carries a client device, walking while keeping natural body orientation, so occasionally her own body blocks the LoS of the AP-client link. We measure the *AoA profile* at the client side, at 11 random locations. The *AoA profile* depicts the received signal strength (RSS) along each angular direction. We omit weak AoAs that are 5 dB lower than the strongest one, because of their negligible contributions to the total RSS. From a high level, the AoA profile shows the multi-path diversity and each path's quality at each specific location. For simplicity, we only use APA at the AP side, whereas the client is tuned to a quasi-omni beam, but the effects can be reciprocal. Fig. 4.1 (a) and (b) show the resulting AoA profiles. Each line segment in the polar plot represents the AoA of a signal path and the segment length denotes the corresponding RSS. We have two major observations from the results.

Limitation of single-array: *A single-array AP has very limited FoV and creates very limited multipath.* As shown in Fig. 4.1 (a), only a few locations within the single array's FoV have reasonable RSS. Beyond those are the AP's blind spots and the client can only rely on the NLoS reflected signals, which tend to be weaker and come up in a sporadic way. Overall, the single-array AP can only provide very few dominant paths for only locations within its FoV. We note that recent work [124] characterized the FoV constraint of 802.11ad radios, but the focus was on the antenna gain pattern, so reflection/blockage effects are not analyzed.

The benefit and potential of APA: *An APA is able to expand the coverage*

dramatically and provide more multipath diversity, potentially leading to more robust links under mobility and blockage. With a 4-array AP, more locations have strong LoS paths owing to the complementary FoVs of multiple arrays, even for those originally in the blind spots of the single-array case. In addition, almost all client locations receive signals from multiple dominant AoAs, *i.e.*, the APA can make better use of surrounding reflectors, since the enlarged FoV contains signals with more diverse AoAs. This benefit exists even for locations whose LOS are blocked. The additional multipath will be valuable for maintaining a robust connection, because the signals can be resteered along a new path's direction even if one is blocked. However, note that for certain client locations (*e.g.*, top left), the APA leads to weaker RSS than the single-AP. This implies that *it may not be optimal to activate all arrays simultaneously*, since the transmit power is spread out, and certain arrays' signals may cancel each other.

4.2.2 Challenges

The foregoing measurement reveals the potential of APA, assuming an oracle system that can orchestrate the arrays and beams with no overhead. In practice, approaching this ideal entails non-trivial design challenges.

Joint array and beam management for mobile users: An 802.11ad AP executes a sector level sweeping (SLS) periodically at the beginning of each beacon interval (BI), where it broadcasts signals sequentially through each of its N beam patterns. The periodic beam scanning ensures the AP is discoverable by unassociated clients, and the best beam can be identified for each associated client. The SLS beam scanning involves N

52-byte beamforming (BF) frames and $N-1$ 1-μs Short Beamforming Inter Frame Spacing (SBIFS), two 208-bit SSW-feedback frame and one Long Beamforming Inter Frame Spacing (LBIFS) [138]. The total scanning overhead is relatively small, *e.g.*, $\sim$1.1 ms for a 64-beam array, in contrast to the typical BI of 100 ms. However, for mobile clients, the *beam coherence time*, *i.e.*, mean period within which the best beam index remains unchanged, becomes much shorter. So the beam scanning has to be executed more frequently. Specific to the APA, the scanning overhead is further multiplied by the number of arrays, rising to $\sim$8.8 ms even for an 8-array AP. Besides, even if we ignore the protocol overhead, the AP has to jointly decide beam selection and array selection (*e.g.*, billions possibilities, Sec. 4.3.3), a combinatorial problem that can easily exhaust its computational power.

At first blush, one can simply turn on all phased arrays to circumvent the array selection problem. Unfortunately, the FCC regulation [139] imposes constraints on both the total radiation power (TRP) and effective isotropic radiation power ($EIRP$). The EIRP constraint limits the phased array gain along the peak direction to 43 dBm and the average of all directions to 40 dBm, for safety and interference concerns. The TRP constraint further limits the total emission power of all directions to 500 mW [139]. When all the arrays on an APA are active, the TRP needs to be split among all of them, whereas an optimal solution should concentrate all the power towards the strongest eigen mode of the channel, *i.e.*, beaming the signals towards the strongest AoA.

Co-phasing between phased arrays: The phased arrays on the same APA node share the same RF chain and transmit the same digital baseband signals. However, due to their relative location/orientation differences, and hardware-induced initial phase offset,

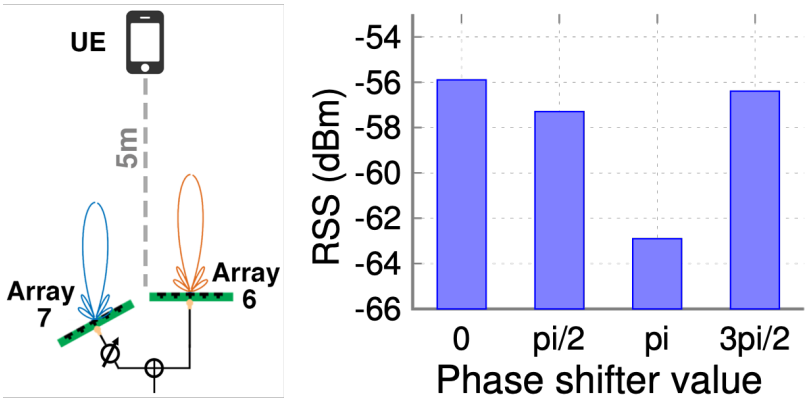


Figure 4.2: The impact of co-phasing.

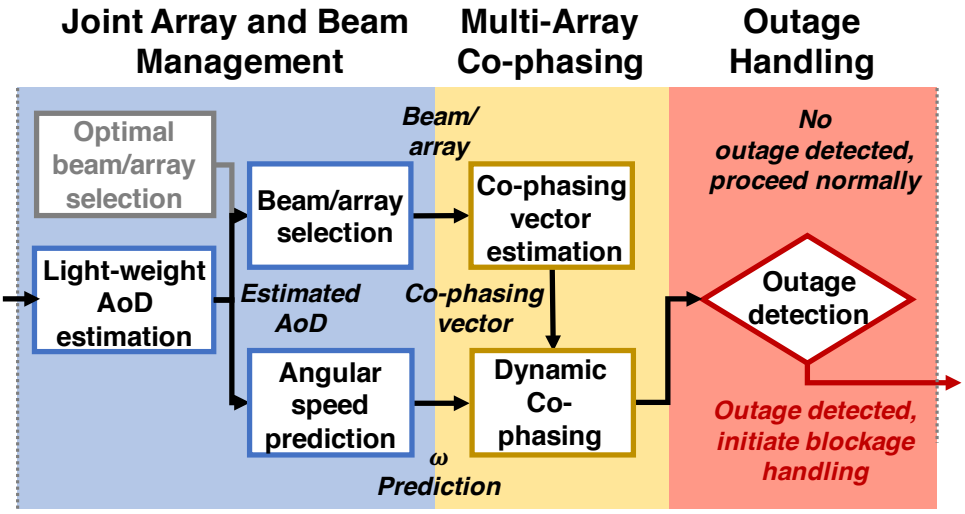


Figure 4.3: X-Array workflow.

the emitted signals do not necessarily combine coherently at the receiver. To showcase this phenomenon, we run a controlled experiment with two arrays (index 6 and 7) on the AP facing the client direction. Fig. 4.2 shows up to 7 dB of variation in RSS, as the relative phase between the two arrays varies between $0, \frac{\pi}{2}, \pi$, and $\frac{3\pi}{2}$. This implies a strong need for phase compensation, or *co-phasing*, to ensure coherent signal combination between concurrent arrays. The problem becomes more pronounced in mobile scenarios, as slight location variation causes significant phase change (due to the short wavelength). The need to choose the optimal co-phasing factor essentially adds one more dimension in the APA's decision space, making it intractable. Note that our 802.11ad radio only allows configuring the co-phasing factor with a 2-bit resolution (4 relative phase values). Finer phase resolution will further compound the decision complexity.

Recovering from link outage, especially under blockage: MmWave link outage may occur in an unpredictable manner, due to other objects moving across the LoS or the device user's own body blockage. For single-array radios, existing work has explored algorithms to realign the transmitter and receiver's beams, taking advantage of the correlation between beam patterns on the same phased array [117, 119]. Yet for an APA, a new mechanism is needed that can reselect the array as well as its optimal beam to leverage the multipath diversity. And again, a brute-force way of rescanning all arrays may incur non-trivial overhead, especially when the user is moving and body blockage occurs frequently.

4.3 Design

4.3.1 Design Overview

We now briefly introduce X-Array's design components and workflow. We build X-Array on top of the 802.11ad MAC/PHY stack. Without loss of generality, we assume a single client served by the X-Array access point. Extension to multiple users can be straightforwardly realized using the built-in MAC protocol in 802.11ad, *i.e.*, transmitting to each client sequentially with CSMA/TDMA based scheduling. We assume the AP uses an APA whereas the mobile client has a single phased array due to form-factor constraint. To ensure it is discoverable by clients facing arbitrary directions, the X-Array AP has to follow the 802.11ad SLS (Sec. 4.2) to periodically broadcast a beacon frame through each beam and repeat it for each array. Considering the overhead of such *full scanning* (Sec. 4.2), it has to be activated infrequently (default to every 8 BIs in X-Array).

X-Array's main design components and decision logic run on the multi-array AP, shown in Fig. 4.3. Whenever a client is associated, the AP runs a one-time full-scanning. The client calls a lightweight *AoD estimation algorithm* (Sec.4.3.3) and feeds back its estimation to the AP. Given the current AoD, the AP uses a lookup table to select the optimal array(s) and beams to activate. The table only needs to be generated once in an offline manner, using a *joint array/beam selection algorithm* (Sec.4.3.3) that optimizes the overall APA beam output pattern with respect to each AoD, under the TRP and EIRP constraints. Afterward, the client and AP proceed to their runtime routine, and periodically update the array/beam selection, based solely on the per-BI SLS beacon

broadcast from one of the activated arrays.

Meanwhile, whenever two or more arrays are activated, X-Array applies a *multi-array co-phasing algorithm* (Sec.4.3.4) to ensure that the arrays' signals are coherently combined. To align the signal phases, X-Array AP estimates the inter-array phase difference, and then compensates the difference by applying an initial phase offset (*i.e.*, co-phasing factor), which is allowable in commercial 802.11ad APA hardware (Sec. 4.4). As will be verified in Sec. 4.3.4, keeping track of the inter-array phase offset directly incurs huge measurement overhead. Therefore, the X-Array AP predicts the phase offset changes within one BI instead, based on the changing rate of the estimated AoD. It continuously applies the co-phasing factor based on the predicted phase offset, until the beginning of the next BI when the AoD is refreshed.

Occasionally, the mmWave link may experience an outage, *i.e.*, low or null RSS, likely due to AP losing track of client under blockage coupled with abrupt motion. Then the AP executes a novel *multi-array concurrent beam scanning* scheme to rediscover strong signal paths and reidentify the best array/beam. This scheme reduces the outage recovery overhead from 8.8 ms to around 1.2 ms on an 8-array node (64 beams per array), compared with a full scan.

4.3.2 Preliminaries: Modeling APA Multi-Array Beamforming

We first introduce a model of APA, which is a basis for the exposition of the X-Array design. For simplicity, we assume a Uniform Linear Array (ULA) with antenna elements arranged along the azimuth plane with half-wavelength displacement. Note that

the design can be easily extended to Uniform Planer Array (UPA), and our implementation uses 6×6 UPA.

Modeling single-array. For a single ULA with N antenna elements, assuming omni-directional Rx, the received signal can be formulated as:

$$\mathbf{y} = \mathbf{H}^T\mathbf{w}\mathbf{x} + \mathbf{n} \tag{4.1}$$

where $\mathbf{H}$ and $\mathbf{w}$ both are 1-by-N vectors, representing the channel gain from the N transmit antennas, and the beamforming weights, respectively. $\mathbf{x}$ represents the transmitted symbol and $\mathbf{n}$ represents the noise.

The separation between antenna elements, d, is usually half wavelength, much shorter than link distance, so it suffices to model the far field. The channel can be decomposed as gain component $\mathbf{A}_G$, which can be approximated to be consistent across antenna elements, and phase component $[e^{j\frac{2\pi nd}{\lambda}sin(\phi)}]^N, n = 0, 1, \ldots, N-1$, where ϕ is Angle-of-Departure(AoD), the angle between the normal line of the phased array panel and receiver wave-front. Plugging this decomposition in Eq. (4.1), we have:

$$\mathbf{y} = \mathbf{w}^T\mathbf{A}_G[e^{j\frac{2\pi nd}{\lambda}sin(\phi)}]^N\mathbf{x} + \mathbf{n} \tag{4.2}$$

where $[.]^N$ represents a vector of size N, and $n = 1, 2, \ldots, N$. The phase component $[e^{j\frac{2\pi nd}{\lambda}sin(\phi)}]^N$ is usually called *steering vector* for phased array beamforming. To maximize the directionality gain towards ϕ, the complex conjugation of the steering vector at ϕ should be used as the *codebook entry* to form a *beam*, *i.e.*,

$$\mathbf{w}_\phi = [e^{-j\frac{2\pi nd}{\lambda}sin(\phi)}]^N \tag{4.3}$$

Each codebook entry constitutes one row in the *codebook* (a matrix). The codebook entries are often designed to steer to R angles that equally partition the FoV. Note that is applicable to other codebook design objectives as well (*e.g.*, minimizing sidelobes [140]).

Modeling APA. Now consider an APA transmitter with P phased arrays. Similar to Eq. (4.1), we have:

$$\mathbf{y} = \mathcal{H}^T\mathbf{w}\mathbf{x} + \mathbf{n} \tag{4.4}$$

where $\mathbf{w}$ and $\mathcal{H}$ are a 1-by-NP beamforming weight vector and a 1-by-NP channel gain vector, for all NP antenna elements on the APA, which can be seen as a new, giant phased array. *Since the phased arrays may be placed in different positions/orientations, not all NP antenna elements follow a ULA layout. So the beamforming equation Eq.* (4.3) *does not necessarily hold for APA*. Alternatively, we can regard the Rx signal as the coherent combination of signals from the Tx arrays. Then Eq. (4.4) can be rewritten as:

$$\mathbf{y} = \sum_{p=0}^{P-1}\mathbf{w}_p^T H_p\mathbf{x} + \mathbf{n} \tag{4.5}$$

where H_p is 1-by-N channel gain vector of the phased array with index p. Since the two phased arrays are co-located on the same device, and typically only separated by several centimeters, shorter than the Tx-Rx distance. Thus, we can again make the far-field assumption, *i.e.*,

$$\mathbf{y} = \sum_{p=0}^{P-1}\mathbf{w}_p^T\mathbf{H}_p e^{j\frac{2\pi s_p}{\lambda}cos(\phi+\frac{\delta_p}{2})}\mathbf{x} + \mathbf{n} \tag{4.6}$$

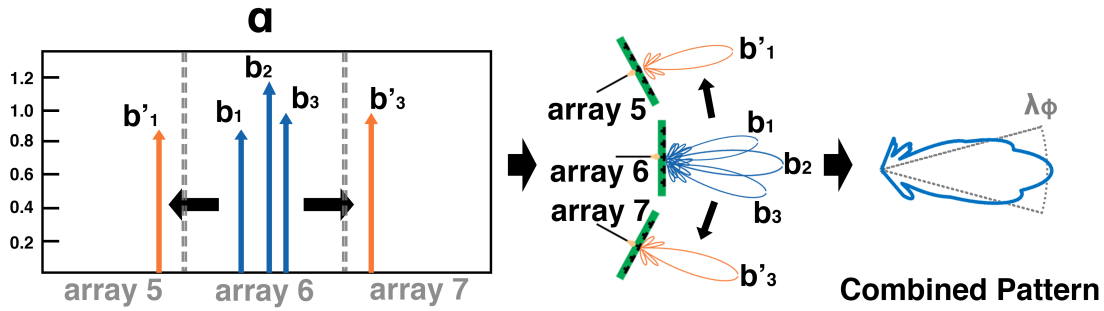


Figure 4.4: X-Array optimization relaxation: replace and redistribute beams to match λ_ϕ.

where s_p is the displacement between the centroids of array p and a reference array, and δ_p denotes the angle between their normal directions. $\mathbf{H}_p$ is the channel gain vector for array p, similar to the term $\mathbf{H}$ in Eq. (4.1) . We define the inter-array phase difference as *co-phasing vector* $\mathbf{E}_\phi = [e^{j\frac{2\pi s_p}{\lambda}cos(\phi+\frac{\delta_p}{2})}]^P, p = 1, 2, \ldots, P$.

4.3.3 Joint Array and Beam Management

Optimal Array/Beam Selection for a Given AoD

We formulate the joint array/beam selection as an offline optimization problem. For a given AoD, the objective is to appropriate the optimal beam pattern to maximize the power towards the AoD direction subject to TRP and EIRP constraints, while creating maximum multipath diversity. Note that the one-time offline optimization solely depends on the line-of-sight AoD. Hence it does not need to rerun when environment changes.

Without loss of generality, we assume the APA jointly covers an FoV of 360° on the azimuth plane where the clients are located. We equally partition the 2π FoV into R

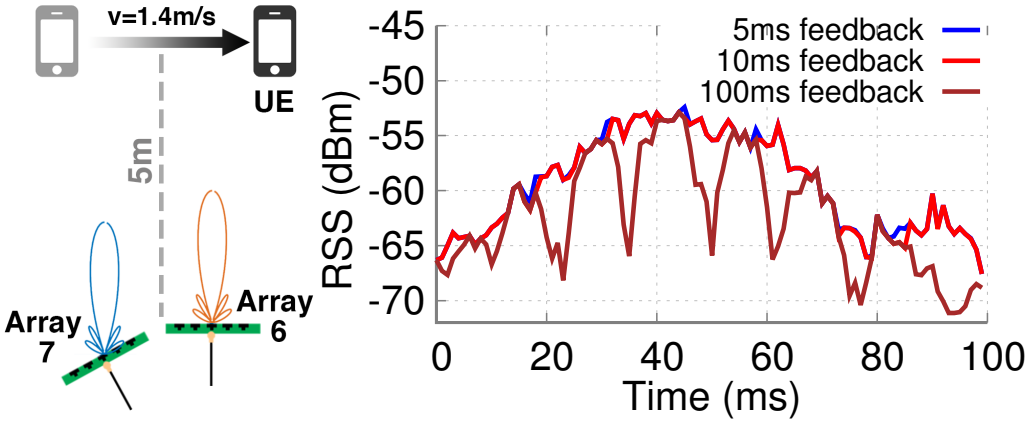


Figure 4.5: Mobility causes two arrays to lose co-phasing periodically, unless with frequent feedback.

directions and denote each partition as $\varphi_r = \frac{2\pi(r-1)}{R}, r = 1, 2, \ldots, R$. The vector of RSS values of all R directions, or normally called "beam pattern", of the beam indexed b on array p, can be represented as:

$$\mathbf{r}_{b,p} = |\mathbf{w}_{b,p}^T [e^{j\frac{2\pi nd}{\lambda} sin(\varphi_r)}]^{N\times R}| \tag{4.7}$$

where $[.]^{N\times R}$ represents a N-by-R matrix; $n = 1, 2, \ldots, N$ and $r = 1, 2, \ldots, R$. Here we omit the channel gain factor A_G in Eq. 4.2 as it contributes equally for all beams. Correspondingly we express the collection of beam patterns on a phased array p as $\bar{\mathbf{R}}_p = \{\mathbf{r}_{b,p}, b \in B\}$. As we mentioned earlier, the phased arrays in X-Array share the same codebook. Hence *the beams of the same index on different arrays share the same beam patterns, although they might point at different directions due to the arrays' orientation differences.*

The optimization problem can be formulated as:

$$\max_{\alpha} \quad \lambda_\phi \sum_{b,p} \alpha_{b,p} \mathbf{r}_{b,p}^T \tag{4.8}$$

$$\text{s.t.} \quad \sum_{b,p \in \varepsilon} \alpha_{b,p} \mathbf{r}_{b,p} \leq I_{\text{EIRP}} \ (elementwise) \tag{4.9}$$

$$|| \sum_{b,p} \alpha_{b,p} \mathbf{r}_{b,p} ||_1^1 \leq I_{\text{TRP}} \tag{4.10}$$

$$\alpha_{b,p} \in \{0, 1\}, \forall b, p \tag{4.11}$$

$$\sum_{b} \alpha_{b,p} \leq 1 \tag{4.12}$$

where λ_ϕ is a 1-by-R vector associated with a given AoD ϕ, and α is a B-by-R binary decision matrix. The λ_ϕ is the ideal APA beam pattern for the given AoD ϕ. For simplicity, we define λ_ϕ as a simple binary vector which has unit gain within $\pm 20\circ$ of its intended AoD and 0 gain elsewhere. Eq. (4.10) and (4.9) are the TRP and ERIP constraints, respectively. The constraints (4.11) and (4.12) represent the fact that we can choose at most one beam on one array. λ_ϕ represents a customized ideal beam pattern. The goal of the maximization function is to find a selection of beams whose combined pattern best matches λ_ϕ. We define λ_ϕ as a simple binary vector where the direction $\pm 15\circ$ around AoD has 1 and others 0. We show in Sec. 4.5 that even with such simple definition, X-Array can substantially improve the beam coherent time (Sec. 4.2.2).

The maximization objective (4.8) essentially combines the RSS values of individual beam patterns as the resulting joint beam pattern of multiple beams on multiple arrays. In other words, we model the average power combination of the beams. Later we will introduce the dynamic co-phasing design (Sec. 4.3.4) which ensures the different beams'

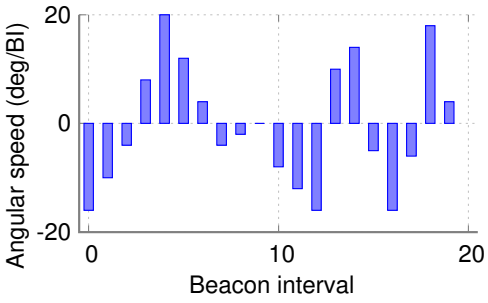


Figure 4.6: Client angular speed.

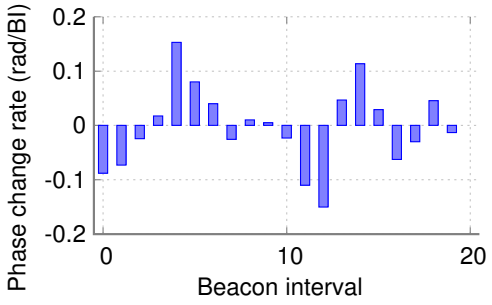


Figure 4.7: Phase changing rate.

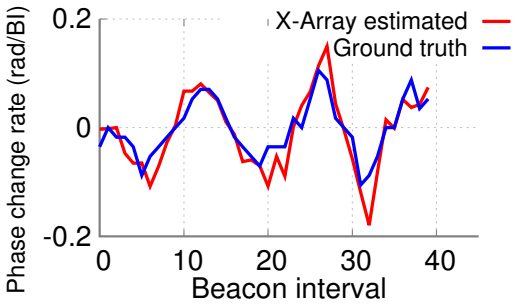


Figure 4.8: X-Array phase prediction matches the ground truth well.

signals are coherently combined at run time to further enhance SNR.

This optimization framework is non-linear, and the search space is determined by the number of beams B and arrays P. A brute-force way of solving the problem requires searching across B^P beam patterns, *i.e.*, 8 arrays with 64 beams on each requires $64^8 \approx 2.8 \times 10^{14}$ times computation which is intractable. Thus, we relax the constraints (4.11) and (4.12). That is, we allow choosing an arbitrary number of beams on one array. Through this relaxation, we transform the previous combinatorial optimization into a standard linear optimization problem:

$$\max_{\alpha} \quad \lambda_\phi \sum_{b,p} \alpha_{b,p} \mathbf{r}_{b,p}^T + \beta \sum_{b,p} \alpha_{b,p} \tag{4.13}$$

$$\text{s.t.} \quad \sum_{b,p \in \varepsilon} \alpha_{b,p} \mathbf{r}_{b,p} \leq I_{\text{EIRP}} \ (elementwise) \tag{4.14}$$

$$|\sum_{b,p} \alpha_{b,p} \mathbf{r}_{b,p}|_1^1 \leq I_{\text{TRP}} \tag{4.15}$$

$$\alpha_{b,p} \geq 0, \forall b, p \tag{4.16}$$

Here we also add the sum of α to the maximization goal in order to encourage the α to have fewer terms. This relaxed version of array/beam optimization can be solved efficiently by standard linear programming toolboxes.

By relaxation, we allow having multiple beams in one array in α which is infeasible in practice. To fix this problem, we leverage a key observation: The FoVs of multiple arrays are usually partially overlapped, so certain beams on different arrays share similar directions. This indicates we can replace the multiple beams on one array with beams of similar directions on adjacent arrays. As a result, to make α feasible, we can simply

identify the multiple beams on one array in α, and replace the extra beams with beams of similar directions on neighboring arrays. Fig. 4.4 illustrates this process. The similarity of directions between arrays is subject to vendor implementation, *i.e.* the size and geometrical layout of phased arrays. Without loss of generality, we use beam AoD to measure the similarity of beams instead of arrays.

In case when there are not enough similar beams to replace, we discard the beams whose directions deviate the most from the AoD, to guarantee the gain along the AoD direction. Note that there may be two reasons for the lack of similar beams: *(i)* The optimization includes too many non-zeros terms in α. *(ii)* The arrays do not have much FoV overlap, so the number of similar beams is small. We prevent *(i)* by adding the regulation term $\beta \sum_{b,p} \alpha_{b,p}$ in Eq. (4.13). On the other hand, *(ii)* can be avoided with more arrays (*e.g.*, when per-array FoV is 120°).

AoD Estimation and Tracking

AoD is assumed as a given input in the aforementioned array/beam selection algorithm. To estimate the AoD, X-Array runs the 802.11ad SLS which scans the beams on one of its currently active arrays. Intuitively, we choose a *primary array* whose orientation is closest to previous AoD, and let the client measure the sequence of *per beam RSS*, denoted as $[r_b]^B$ where b is beam index. According to Eq. (4.2):

$$r_b = |\mathbf{w}_b^T [e^{j\frac{2\pi n d}{\lambda} sin(\phi)}]^N + \mathbf{n}| \tag{4.17}$$

Since the SLS beam scanning is very short (Sec. 4.2), the AoD can be assumed relatively

stable during SLS. *Our objective here is to estimate the AoD ϕ based on the $[r_b]^B$ measurement.* Since the codebook weights $\mathbf{w}_b$ and element-spacing d are known, we can compute the $[r_b]^B$ for any given ϕ when omitting noise. The $[r_b]^B$ that best correlates with the measurement should correspond to the most likely ϕ. We formalize this intuition through a matched filter design.

The matched filter is a B-by-R matrix derived from the product of two parts: *(i)* $[\mathbf{w}_b]^{B\times N}$ is the beam weights of B beams arranged in B-by-N matrix; *(ii)* $[e^{j\frac{2\pi nd}{\lambda}sin(\phi_r)}]^{N\times R}$ is normalized steering vector of R directions arranged in N-by-R. As an extension to Eq. (4.17), the dot product of *(i)* and *(ii)* produces the RSS sequences for all R possible AoDs. Then the matched filter simply correlates itself with the measured RSS sequence $[r_b]^B$ as follows:

$$\nabla = [r_b]^B[\mathbf{w}_b]^{B\times N}\big([e^{j\frac{2\pi nd}{\lambda}sin(\phi_r)}]^{N\times R}\big)^T \tag{4.18}$$

Then we can estimate AoD by identifying the direction r that leads to the maximum similarity between measurement and model:

$$\phi = {}_{r\in R}\,\nabla \tag{4.19}$$

X-Array's AoD estimation mechanism leverages the legacy SLS beam scanning in 802.11ad (Sec. 4.2), so it requires no hardware modification and shares the same overhead (*e.g.*, ∼1.1 ms out of each BI of around 100 ms). Many existing systems [118, 119, 141] also need AoD as an input, but they either lack support on commodity hardware (due to needs for CSI), or they require non-trivial computational time.

It is tempting to think that one can detour the array selection problem, by treating the APA as a *single giant phased array* and apply a single codebook to it. However, this *single giant array approach* lacks scalability for two reasons: *(i) Hardware constraints.* The single codebook needs to specify all possible array/beam combinations, which easily reaches billion scale as mentioned above, way beyond the storage capability of on-board memory (only several hundred KB on a typical 802.11 device[133]). *(ii) Protocol overhead.* Scanning through all the entries on the giant codebook takes 4.8×10^9 seconds for the typical 8-array APA, and will obviously hinder normal data transmissions.

4.3.4 Multi-Array Co-Phasing

X-Array's co-phasing design aims to maximize the beamforming gain when multiple arrays are activated. To overcome the challenge of phase sensitivity (Sec. 4.2.2), we propose a novel dynamic co-phasing scheme that approximates the optimal coherent combination of multiple arrays, without the overhead of constantly probing their phase offsets.

Decomposition

Recall the multi-array channel can be modeled as a composition of signals from individual arrays through a shared channel with phase offsets (Eq. (4.6)). To ensure coherent signal combining, the beamforming weights $\mathbf{w}^T$ must be designed to compensate for the different arrays' phase offsets. Based on Eq. (4.6), we thus have:

$$\mathbf{w}^T = \sum_{p=0}^{P-1} \mathbf{w}_p^T e^{-j\frac{2\pi s_p}{\lambda} cos(\phi + \frac{\delta_p}{2})} \tag{4.20}$$

Recall $\mathbf{E}_\phi = [e^{j\frac{2\pi s_p}{\lambda}cos(\phi+\frac{\delta_p}{2})}]^P, p = 1, 2, \ldots, P$, represents the inter-array phase differences, or *co-phasing vector*. We can then rewrite the APA beam weights equation Eq. (4.20) as:

$$\mathbf{w}^T = w_P^T \mathbf{E}_\phi^* \tag{4.21}$$

where $(\cdot)^*$ is the complex conjugate operator. This implies that *the multi-array co-phasing problem can be decomposed as beamforming on individual arrays, but with proper inter-array phase alignment*. Therefore, to realize co-phasing, we do not need to modify the existing codebook. Instead, we can simply multiply each individual array's codebook entry with an offset $[\mathbf{E}_\phi^*]_p$, which is allowable on commodity hardware (Sec. 4.4). We now describe how to estimate the $\mathbf{E}_\phi$.

Estimating Co-Phasing Vector

We adopt a measurement driven method to estimate the $\mathbf{E}_\phi$ at the beginning of a BI. Specifically, after each individual array's beam is determined (represented by $\mathbf{w}_p^T$) on each array p, we regard one of the arrays as *reference array* with phase 0. To measure the relative phase of other currently active arrays (denoted as *side arrays*), the AP transmits 4 BF frames (the reference signal used in 802.11ad) using the reference array and one side array *simultaneously*. We apply a phase shift $e^{j\varphi}$ to each of the BF frames. Specific to our 802.11ad radio, $\varphi \in \{1, j, -1, -j\}$ (Sec. 4.4). The RSS value for these BF frames can be formulated as:

$$y_\varphi = |\mathbf{w}_p^T \mathbf{H}_p e^{j\frac{2\pi s_p}{\lambda} cos(\phi + \frac{\delta_p}{2})} e^{j\varphi} + \mathbf{w}_0^T \mathbf{H}_p|, \varphi \in \{1, j, -1, -j\}$$

By applying the phase shift value, we essentially build a discrete Fourier series with $e^{j\varphi}$ as "frequency basis" and the co-phasing vector as "coefficients". Hence, to extract the co-phasing vector, we only need to take a Fourier transform on this series:

$$\mathbf{d}^{|\Phi|} = \mathbf{FFT}(\{y_1, y_j, y_{-1}, y_{-j}\}) \tag{4.22}$$

Then we find the second term $d_{p,2} = e^{j\frac{2\pi s_p}{\lambda} cos(\phi + \frac{\delta_p}{2})}$. Recall that the co-phasing vector (Sec. 4.3.2) is the inter-array phase difference between reference array and another side array. For array p, this phase offset is exactly $d_{p,2}$. Hence we repeat this process for all currently active arrays and we have co-phasing vector:

$$\mathbf{E}_\phi = [d_{p,2}]^P \tag{4.23}$$

We need 4 BF frames (each lasting 0.015 ms) for every active array except the reference array. Hence, the co-phasing vector measurement for one array takes 0.06ms. Even with 8 active arrays, the total overhead is negligible (< 0.5 ms).

Dynamic Co-Phasing

The foregoing co-phasing algorithm assumes that the AoD information is always available as input. However, the inter-array phase offset varies drastically over a few ms under node mobility, whereas the AoD can only be updated per BI in order to tame the estimation overhead (Sec. 4.3.3).

More specifically, recall that the steering vector in Eq. (4.2) has a changing rate of $\frac{2\pi nd}{\lambda}sin(\phi)$ and the co-phasing vector, as shown in Eq. (4.6), has a changing rate of $\frac{2\pi s_p}{\lambda}cos(\phi + \frac{\delta_p}{2})$. The array displacement s_p is significantly larger than d, the antenna element spacing. This implies that, with the same angular movement ϕ of Rx, *the APA Tx (affected by both fast-changing co-phasing vector and steering vector) suffers more from link degradation than the single array Tx (affected only by mild steering vector).*

Intuitively, one can keep track of the array steering term by probing the AoD and update the co-phasing factor more frequently. However, the small coherence time of the array steering term requires an impractically high feedback frequency to prevent link degradation. As a showcase, we activate two beams on two arrays on an AP, and move the client at walking speed. We repeat the experiment with different phase feedback intervals (100 ms, 20 ms, 5 ms). Fig. 4.5 shows that the throughput converges only when feedback interval is shorter than 10 ms. Larger intervals cause sub-optimal throughput "valleys" due to laggy feedback. At higher moving speed, even more frequent feedback is needed.

We address this challenge using an *angular speed based dynamic co-phasing* scheme. *Dynamic co-phasing* obtains the fine-grained phase estimates within the scope of one BI, *i.e.* between two consecutive phase feedbacks. At the beginning of a BI, the AP measures the co-phasing vectors for the currently active arrays as the initial co-phasing vector $\mathbf{E}_{\phi,t_0}$. Within each BI duration, to avoid the feedback, we predict the optimal instantaneous co-phasing vector by modeling the relationship between the array steering vector and angular speed. Denote $\Delta\tau$ as AoD estimation interval, at a given time t_j when AoD is updated, we can estimate the average angular speed of the client as:

$$\bar{\omega_j} = \frac{\phi(t_j) - \phi(t_j - \Delta t_\tau)}{\Delta\tau} \tag{4.24}$$

We assume the angular speed is stable within $\Delta\tau$ since a typical BI is very short. Thus, we can predict the phase change within one $\Delta\tau$ interval as a function of time:

$$\frac{2\pi s_p}{\lambda} cos(\bar{\omega_j}\tau + \phi(t_j) + \frac{\delta_p}{2}) \approx \text{ang}(\mathbf{E}_{\phi,t_0}) + \bar{\omega_j}\tau \tag{4.25}$$

To demonstrate the effectiveness of dynamic co-phasing, we leverage the same experimental setup as in Fig. 4.5, and keep measuring the co-phasing vector every 5ms to get fine-grained ground truth. Meanwhile, we estimate AoD every 100 ms (one BI) and calculate client angular speed by Eq. (4.24). Fig. 4.6 and Fig. 4.7 show the client angular speed and the phase change rate of the co-phasing vector $\mathbf{E}_\phi$. The strong resemblance of these two figures further corroborates Eq. (4.25). We then input the angular speed to our dynamic co-phasing model, and predict the co-phasing vector change over every 5 ms within the next BI. Fig .4.8 shows that the predicted phase changing rate matches the ground truth near perfectly. With instantaneous co-phasing vector known, the AP now can align the phase of arrays within a BI without any explicit client feedback.

4.3.5 Recovering from Link Outage

When blockage occurs, if the LoS still delivers strong RSS or there exists any strong reflection path within the FoV of active array(s), the array/beam management and

co-phasing solutions are still applicable (based on the periodic AoD estimation on the primary array). But if a link outage occurs, *i.e.*, weak or null RSS on the current link, then X-Array invokes its outage handling mechanism. A straightforward way is to repeat beam-sweeping on all arrays and select the strongest beam, but apparently this will incur huge overhead.

To tame the rescanning overhead, we propose a simple *concurrent beam sweeping* scheme. Immediately upon outage, X-Array concurrently beacons a reference frame through the same beam index on all arrays, and repeats this for each beam index within the codebook (shared by all arrays). If any strong LoS/NLoS path exists, then at least one beam will lead to a strong AoD peak. If the strong path falls in the FoVs of multiple arrays, then each such array will have one beam with similar RSS as the peak. But the beam indices tend to differ due to the arrays' different orientations (*i.e.*, the same beam index on different arrays points to different directions).

To showcase this phenomenon, we fix an 8-array AP and put a client 3m away, and then snapshot the per-beam RSS of concurrent beam sweeping when a human blocks the LoS path, and after the blockage exits. Fig. 4.9 (a) and (b) plot the results. It can be seen that, upon blockage, two weak peaks exist on the per-beam RSS sequence, likely because a certain beam can establish a NLoS path. Whereas after blockage disappears, three strong peaks reappear.

If the peak RSS after concurrent scanning still falls below the threshold for the minimum bit-rate, then no array/beam can sustain the link, and extraneous connectivity solution may be needed (*e.g.*, [142]). Otherwise, upon confirming the existence of a usable

beam, X-Array needs to further discriminate which array(s) cover the LoS within its FoV. The concurrent scanning result already indicates the best beam indices that lead to strong AoD. So X-Array simply sends a reference frame through the corresponding beam index on each array. Those arrays and beams that lead to AoD peaks will be selected as active arrays. Afterwards, X-Array moves out of the outage mode and transitions into its normal mode of operation. Overall, the concurrent scanning mechanism can be called on to reidentify a strong beam when the current link's RSS drops significantly. When a link is under blockage, it can also be called periodically to check whether the blockage disappears and a new strong path reappears.

Two additional issues are remarkable here: *(i)* A straightforward full-scan needs to probe NB beams in total, vs. $(N + B)$ with concurrent beam sweeping. On an 8-array APA with 64 beams per array, this means the latter reduces the overhead of rediscovering strong paths by $\sim 8\times$ (8.8 ms vs. 1.2 ms). *(ii)* Although co-phasing may sometimes weaken the strongest peak on the RSS sequence, it rarely removes the peak, because the strengthening effects may show up on other beams pointing close to the AoD. Plus, we only need to know *whether* a strong peak exists, so a coarse grained per-beam RSS sequence suffices.

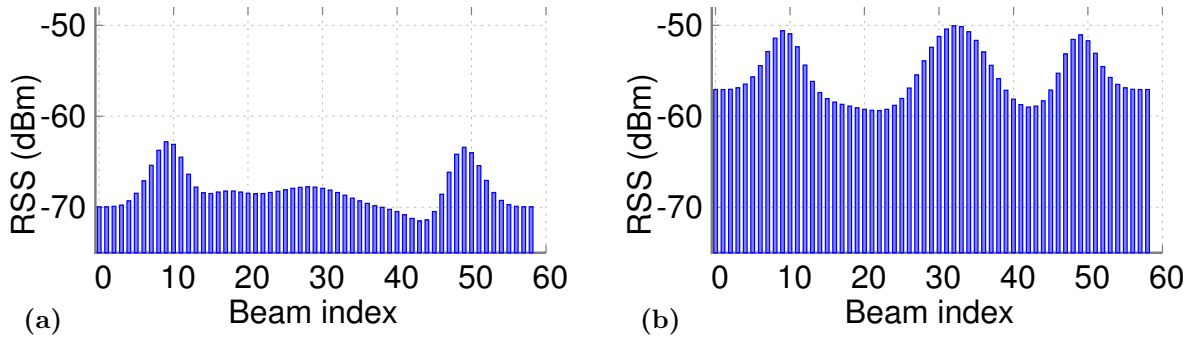


Figure 4.9: Concurrent beam sweeping when: (a) blockage occurs; (b) blockage disappears.

4.4 Implementation and Experiment setup

4.4.1 Implementation

We implement X-Array based on an off-the-shelf 802.11ad AP from Airfide Inc. [126]. The original Airfide AP puts 8 phased arrays on a plane. We reorganize its physical layout, and 3D print an antenna stand (Fig. 4.10) so that the 8 arrays face different azimuth directions with 45◦ separation. In this way, the FoVs of different arrays partially overlap, and together they cover 360° azimuth and 120° elevation. The mobile client has a single phased array. Both devices comprise an 802.11ad NIC (with Qualcomm QCA9500 FullMAC WiGig chip and QCA6335 baseband) with 2-bit phase shifters, plugged in an embedded Linux host (running Wil6210 firmware and driver). Below we describe the notable technical thrusts in implementing X-Array.

(1) *Fine-grained per beam RSS extraction*: The per beam RSS at client side is crucial

for the AoD estimation and co-phasing mechanism, but is concealed to upper layers on commodity 802.11ad devices. To expose the RSS, we first disassemble the firmware file to ARC assembly code. Then we blanket search the assembly code and pinpoint where the per beam RSS calculation takes place, and the associated memory address in NIC. Then we leverage Talon-tools [143], a C-based firmware patching framework adapted for the 802.11 radios, and write the firmware patch to copy the RSS value, immediately after a RSS value is calculated, from its original address to a designated memory address at the very back of NIC memory, which can be safely accessed by the host driver. With the patched firmware loaded to the NIC, we then write a Python program to call the *mem_dump* command on the host, and dump the RSS value from NIC to user space.

(2) *Enabling short BI*: Recall that 802.11ad performs SLS beam sweeping per BI, so a shorter BI may make the beam selection more responsive but at larger overhead. X-Array does not require short BI thanks to the dynamic co-phasing design (Sec. 4.3.4). However, to obtain the fine-grained ground truth phase measurement (Sec. 4.5), we need BI as short as possible. The standard 802.11ad radio limits the smallest BI to be 20 ms, which is not enough for this purpose (c.f. Sec.4.3.4). To overcome this barrier, we follow similar steps as above, to disassemble the firmware and pin-point the BI value memory address. Then we hard code the BI value to the firmware patch. Extremely small BI will lead to inaccurate RSS measurement or even firmware crash. We empirically found 5 ms to be the smallest safe value.

(3) *Real-time codebook loading and inter-array co-phasing*: We implement X-Array's dynamic co-phasing by loading selected beams with different phase shifts after AoD es-

timation, and selecting the optimal phase shift based on the dynamic co-phasing design. The 802.11ad standard limits the codebook size (maximum number of beams) to 128. Yet this X-Array implementation requires more than 128 beams (with different phase shift combinations) due to the APA setup. Hence it is necessary to load the codebook at run-time. Normally the codebook file is only loaded from user space to NIC when the interface boots. We use the *HWD_RFC_WRITE_SECTOR* command (0x900, ut_subtype_id: 0x514) in Qualcomm wil6210 driver to write the new codebook to NIC. Then we call *WMI_SET_RF_SECTOR_PARAMS_CMDID* command (0x9A1), which is originally designed to change one entry in codebook, but it can also trigger codebook reloading onto the phased arrays. This way we can change the codebook without rebooting the NIC.

(4) *Enforcing TRP/EIRP constraints.* We enforce the TRP/EIRP constraint by regulating the beams, which is a common practice by COTS phased array devices. Specifically, we enforce the constraints when optimizing the overall APA beam pattern in respect of AoD, as shown in Eq.4.9 and Eq4.10. We then solve this optimization problem for each AoD, so that the optimization output beam pattern complies with the EIRP and TRP constraints. In our real-time implementation, we use the pre-calculated beam and array combination, which automatically enforces the constraints.

(5) *Implementation of other components.* We implement the X-Array AoD estimation algorithm on the client side, where the per-beam RSS measurements are performed. The client then feeds back the estimated AoD along with estimated angular speed to the AP in a single 802.11ad packet. The AP acts accordingly, by calling other design components (implemented as python modules on the user space). Thanks to the lightweight

X-Array design, the algorithms can run in *real-time* on the embedded PCs of both the AP and client.

4.4.2 Experimental setup

For comparison, we have also implemented the following APA solutions [144, 145, 141] as baselines.

(i) ACO [141]: We implemented the phase estimation and AoD estimation algorithms proposed in ACO [141]. Just like X-Array, ACO takes per-beam RSS as input, but it can estimate the phase difference between one reference antenna element and all other antenna elements, from which it obtains the CSI. The corresponding AoD is obtained by running the MUSIC [146] over the CSI. To implement ACO, we generate a custom codebook file with two antennas activated for each beam, corresponding to the reference antenna element and the to-be-measured antenna element. We choose the antenna element with index 0 in the codebook file as reference. We generate 4 beams for each to-be-measured antenna with its phase index as 0, 1, 2 and 3 (mapped to $0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}$). We then load the codebook and feed the per beam RSS to the ACO model [141] which is implemented in python.

(ii) Periodic probing: We also implemented a periodic probing mechanism to realize co-phasing across arrays. A probing frequency of $\frac{1}{2}$ BI means that co-phasing is done twice per BI. Here the inter-array co-phasing is then estimated based on the ACO's CSI output. Higher feedback frequency may make the co-phasing more accurate under mobility/blockage, at the cost of higher overhead.

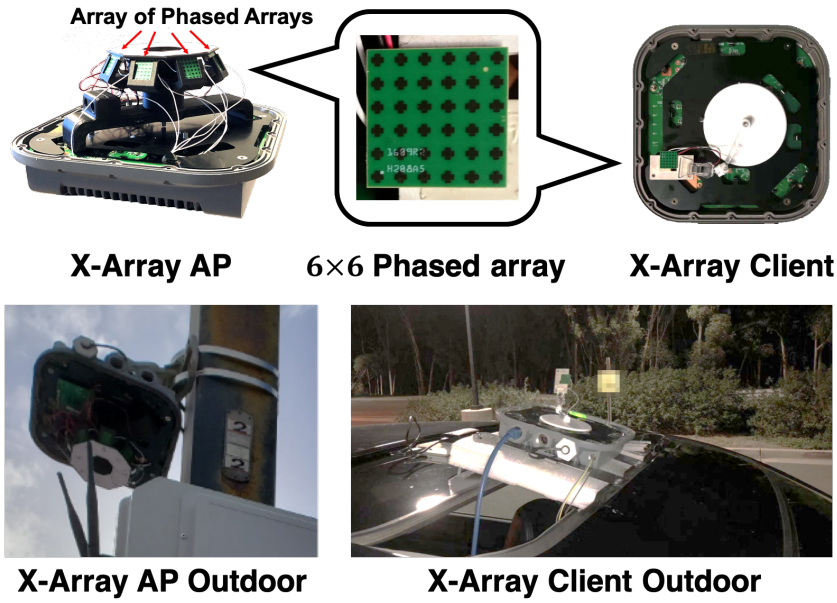


Figure 4.10: X-Array hardware prototype is built on a commercial multi-array 802.11ad AP, with customized array layout.

(iii) Neighbor scan: Since there exists no other work in outage recovery with APA, we implement a *neighbor scan* (NS) baseline. Whenever an outage occurs, NS first scans adjacent arrays with the smallest angle displacement to the previous AoD. If a strong beam exists, it settles on these arrays; otherwise it keeps trying others.

Note that the above *periodic probing*, and *neighbour scan* essentially represent the default behaviors of the existing 802.11 ad protocol when running on an APA radio.

4.5 Evaluation

We conduct extensive experiments in three types of environment settings: indoor open area ($16.6m \times 6.4m$ yoga room) with no furniture blockage, complex indoor (typ-

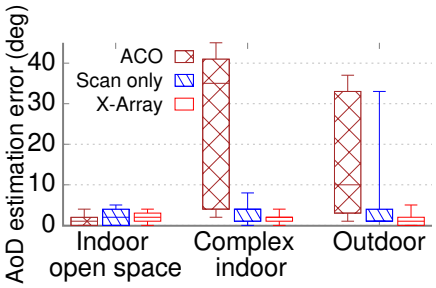


Figure 4.11: AoD estimation error.

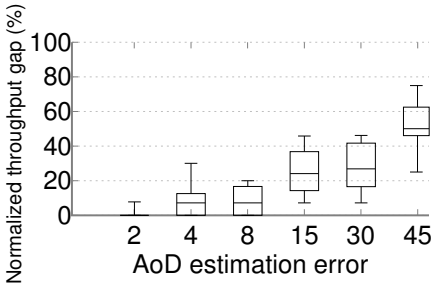


Figure 4.12: Impact of AoD estimation error.

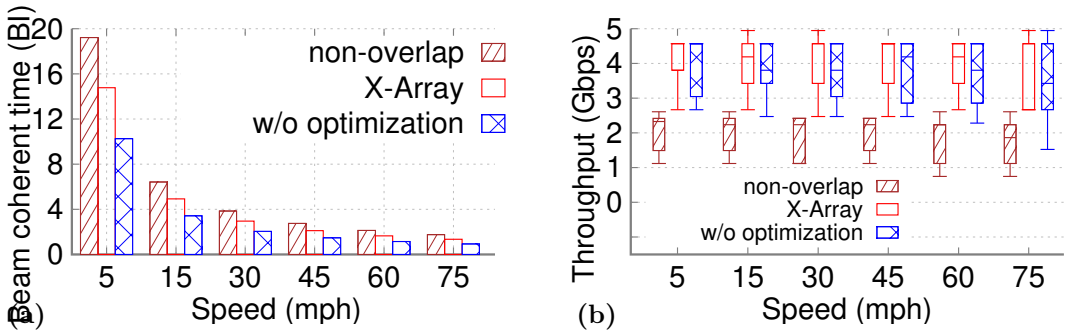


Figure 4.13: Impact of joint array/beam selection on: (a) link stability; (b) link quality.

ical office environment with workbenches and partitions around the route) and outdoor (parking lot). We will verify that X-Array achieves WiFi-like coverage and maintains high performance under link dynamics. Our results can be summarized as follows:

- X-Array maintains high accuracy in AoD estimation across different environments, enabling a negligible throughput gap (around 5%) in comparison with an ideal solution that knows the ground-truth AoD. Also, X-Array's joint beam-array optimization can form wider beams without sacrificing the gain, so it can provide stable and high throughput under high mobility scenarios.

- When taking into account co-phasing, X-Array can more accurately predict the phase change of the client within next beacon interval. Consequently, X-Array leads to a much lower throughput gap (<12 %) in comparison to periodic probing (18% to 42%).

- The blockage recovering mechanism of X-Array saves most of the link outages and experiences no throughput gap in 96% and 93% of the blockage cases, in complex indoor and indoor open space respectively.

- We put all the components together and run X-Array, and we find that X-Array does not suffer any throughput gap most of the time under different speeds, while periodic feedback mechanism has a significant throughput loss even with a high frequency of every $\frac{1}{3}$ beacon interval.

- By maximizing the benefits from APA, X-Array can achieve WiFi-like omni-directional coverage. In comparison, a straightforward way of turning on 4 phased arrays may lead to even worse coverage compared with 2 arrays.

4.5.1 Micro-benchmarks

Joint Array and Beam Management

Accuracy of AoD estimation. We validate the accuracy and effectiveness of X-Array's AoD estimation in three different environments. In each environment, we fix the AP and randomly place the client to 80 locations. We compare three schemes: ACO

[141], X-Array with and without match filter (the latter referred to as "beam scan only"). The box plot in Fig. 4.11 shows that X-Array has the lowest average estimation error and lowest std., *i.e.*, it achieves the most accurate and stable AoD estimation. Although ACO performs slightly better in indoor open space, its estimation error increases dramatically in complex indoor and outdoor scenarios, since its CSI estimation only works reliably under high SNR and degrades a lot in relatively long-range and multipath-rich conditions. Also note that the beam scan only approach performs worse than X-Array in all cases due to the imperfect beam patterns, which implies the effectiveness and necessity of the matched filter design (Sec. 4.3.3).

Impact of AoD estimation error. To understand the end-effect on throughput, we fix the AoD estimation error to a specific value. To control the error, we first measure the groundtruth AoD using a laser range finder. Then we intentionally use an AoD value that deviates from the ground-truth by 2° to 45°, as input to X-Array's array/beam selection. Note that the commodity 802.11ad device does not allow data transmission under RSS monitoring mode. Moreover, it does not implement high bit-rate 802.11ad modulation and coding, so the benefit of higher channel/link quality cannot be reflected in measured throughput. We thus follow the same approach as in [141, 117] to map the RSS to achievable throughput.

Fig. 4.12 plots the percentage of throughput loss compared to the ground-truth, denoted as *Normalized throughput gap*. An AoD error of below 5° causes minor throughput loss (*e.g.*, median 10% and 75-percentile at 15% in 4° case), yet the median throughput loss escalates to about 30% for AoD error above 30°. Considering the AoD estimation

accuracy (Fig. 4.11), the corresponding average throughput loss of X-Array falls below 5%, in comparison to the 20%-40% loss of ACO, in complex indoor and outdoor environment.

Effectiveness of X-Array's joint array/beam selection. We compare X-Array with two baseline approaches, one is to simply lets two arrays beamform to the same direction to emulate the effect of treating the APA as a single giant phased array mentioned in Sec. 4.3.3, denoted as *w/o optimization*, and the other one is to place multiple arrays with non-overlapping FoV, denoted as *non-overlap*. To verify X-Array's resilience under mobility, we use *beam coherence time* (Sec. 4.2) as the performance metric. To create different client moving speeds, we use a time-lapse approach as in [117]: We move the client 2.2cm each time along a 10m trajectory, and collect per-beam RSS traces at each point. Then, different speeds correspond to different time-lapse values between the measurement points. The result in Fig. 4.13 (a) shows that, even at 75 mph moving speed, the beam coherence time of X-Array remains within one BI (100 ms) and non-overlap remains 1.3 BI (130 ms), whereas the giant array can only support up to 45 mph. The reason is that X-Array creates wider beams without sacrificing directionality. Thus, it needs to update the array/beam selection much less frequently, leading to even lower overhead than the giant array approach.

To ensure this benefit does not come at the cost of lower beam quality, we place the client to random positions along trajectory 10 m away, and compare the *average throughput* of the two beamforming mechanisms. The result is shown in Fig. 4.13 (b). Fig. 4.13 (b) shows that, compared with w/o optimization, beams formed by X-Array can achieve comparable or higher average throughput, and lower variance in general. Interestingly, the

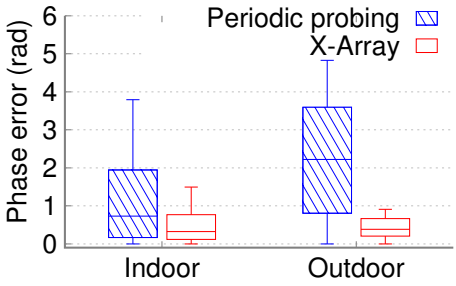


Figure 4.14: Phase prediction error.

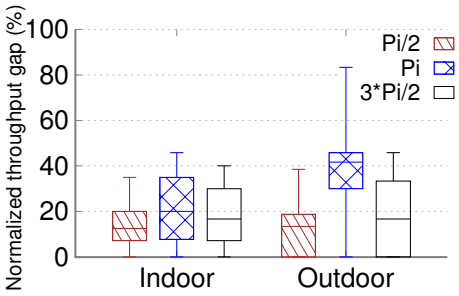


Figure 4.15: Impact of phase prediction error.

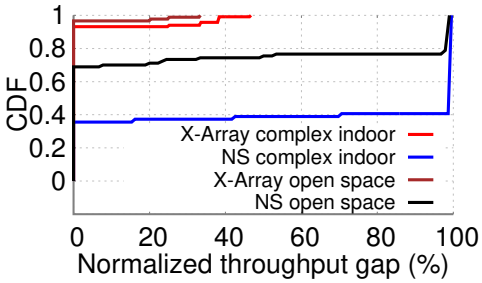


Figure 4.16: CDF of normalized throughput gap under blockage

throughput of both X-Array and w/o optimization is $\sim 2\times$ higher than non-overlap, which indicates that high link quality can be achieved by applying the co-phasing algorithm.

Multi-Array Co-Phasing

Accuracy of phase prediction. We conduct the co-phasing experiments by arbitrary walking along 10 routes inside an office building, and repeat the aforementioned pointwise measurements to emulate different driving speed in outdoor open space. For each beam combination (with two different beam indices on two arrays) at each measurement point, we exhaustively vary the 4 possible co-phasing values between two arrays, and

measure the per-beam RSS when both arrays are activated. As we previously observed in Fig. 4.8, the phase change rate estimated by X-Array shares a highly similar pattern with the groundtruth. According to the predicted phase change rate, we can predict the two arrays' phase offset across the duration of one BI. The box plot in Fig. 4.14 further shows the percentile errors when we run phase prediction in indoor (3 mph walking speed) and outdoor (varying speed from 15 mph to 75 mph) settings. We see that X-Array has an average phase error of 0.57 and 0.44 in outdoor and indoor, which are around 2.5× and 5× lower than periodic probing. Moreover, periodic probing incurs around 2× larger phase error in outdoor scenarios than indoor, as it is unable to track the phase change under high speed. In contrast, X-Array becomes even more accurate when it comes to outdoor. The reason is that although the phase change is fast, it is also stable and thus easier to predict.

Impact of phase prediction error. We now evaluate the impact of phase error in terms of the throughput loss compared with an oracle solution. The groundtruth phase offset between two beams (on two arrays) in the oracle solution is obtained using the method in ACO [141]. As we mentioned in Sec. 4.4, our radio only has a 2-bit phase shifter, which makes the measured impact of phase error at least the impact of 90 degrees phase difference. So we intentionally deviate the predicted phase from ground-truth by $\frac{\pi}{2}$, π and $\frac{3\pi}{2}$. Fig. 4.15 plots the normalized throughput gap generated different deviations. Our previous experiment already showed that X-Array can predict phase accurately with a median phase error of only about 0.5 in radius. This is much lower than the case with phase error of $\pi/2$ which only causes about a median normalized throughput gap of 15%.

Since 0.5 ¡ $\frac{\pi}{2}$, it is clear that X-Array suffers much less normalized throughput gap (around 12%) compared to the $\frac{\pi}{2}$ phase error case. In contrast, recall periodic probing has a phase prediction error between $\pi/2$ to π (Fig. 4.14), this translates into a significant median normalized throughput loss of 18% to 42%. This result will be further corroborated in the system level evaluation (Sec. 4.5.2). Note that higher throughput loss exhibits in outdoor scenario than indoor, because there are fewer reflectors in outdoor scenario which may cause less NLoS to compensate the throughput gap.

Recovering From link Outage

We compare X-Array with the *neighbor scan only* (NS) approach to check its effectiveness and overhead in recovering from blockage. Our experiment investigates indoor scenarios where blockage often occurs due to human activities. We collect 10 trials with random client locations in each environment and evaluate the *normalized throughput gap* when blockage occurs. Fig. 4.16 plots the CDF across all the experiment trials. It is observed that X-Array experiences almost no throughput gap in 96% and 93% of the time in complex indoor and indoor open space, respectively. In contrast, NS suffers from throughput loss in over 60% of the cases, which can be up to 99.8% in complex indoor scenario. The indoor open space does not have as frequent sudden change as complex indoor scenario, thus more likely to find the strongest path in a shorter time, but the 80% normalized throughput gap is still large (about 99%).

4.5.2 System Level Evaluation

Overhead reduction. We compare the overall throughput of X-Array and periodic probing in open space and complex indoor/outdoor environment and conduct the experiments in 5 trials at different locations for each setting. The periodic probing scheme shares the same array/beam selection mechanism as X-Array, but aligns the phase based on different CSI feedback frequencies, *i.e.*, every 1, $\frac{1}{2}$ and $\frac{1}{3}$ BI. Also the high speed results are obtained by predicting phase by X-Array and periodic probing using the method in Sec. 4.5.1 first, measuring the RSS of joint-panel beamforming with the predicted phases, and then mapping it to throughput. The box plot in Fig. 4.17 shows that X-Array has a near-zero median and 75-percentile throughput gap, and only several outliers in all the settings. In contrast, periodic probing suffers from around 20% to 57% normalized throughput gap. Interestingly, although high frequency periodic probing (*e.g.*, at $\frac{1}{3}$-BI intervals) may achieve low normalized throughput gap at low moving speed, the normalized throughput gap increases dramatically when it comes to high-speed case. This is because the phase change rate may easily exceed the probing frequency under high mobility. The experiment verifies the importance of the predictive co-phasing of X-Array under different moving speeds.

Coverage improvement. To verify whether X-Array effectively exploits the coverage advantages of APA (Sec. 4.2), we conduct experiments with different number of arrays and by disabling/enabling its design components in both typical indoor and outdoor scenarios. For indoor scenario, we place the client at certain locations and conduct

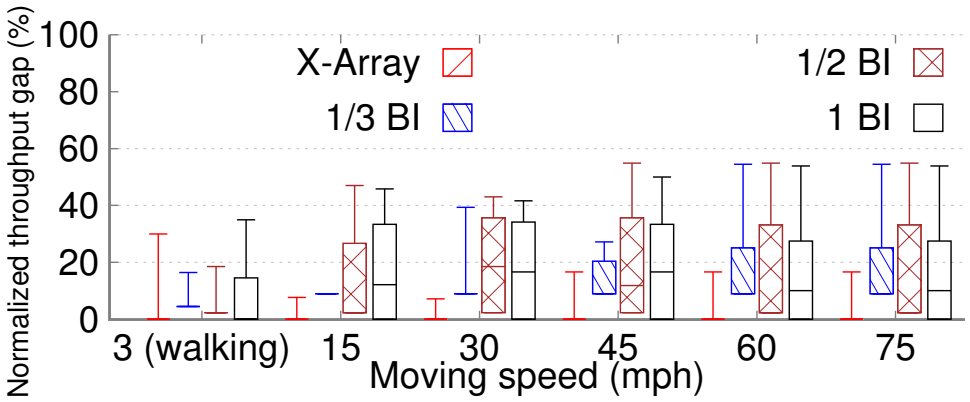


Figure 4.17: Normalized throughput gap due to co-phasing overhead.

measurements across the whole room. For outdoor scenario, we fix the AP on a lamppost (Fig. 4.10) and place the client at different distances, and repeat each distance setting in five different outdoor environments. We evaluate the cases with 2, 4, and 8 arrays running X-Array, and a case with 4 arrays but disabling the array selection and co-phasing (labeled as "4 arrays w/o switching"). According to the specification of our device, the TRP regulation constraint mandates that at most 2 arrays be turned on with full power. So when $N > 2$ arrays are active simultaneously, we reduce the transmit power per array to $2/N$. The result of indoor and outdoor scenarios are shown in Fig. 4.18 and Fig. 4.19, from which we can derive the following major insights: *(i)*. Although 8-array achieves similar coverage as 4-array, the areas with high bit-rate links is much larger, thanks to arrays with partially overlapping FoVs providing co-phasing gains. Overall, with 8-arrays, X-Array eliminates all blind spots in the room, even for locations with thick walls, *i.e.*, it approximately achieves omni-directional coverage. *(ii)*. In the indoor and some outdoor cases, the 4-array w/o switching performs even worse than 2-array running X-Array. This

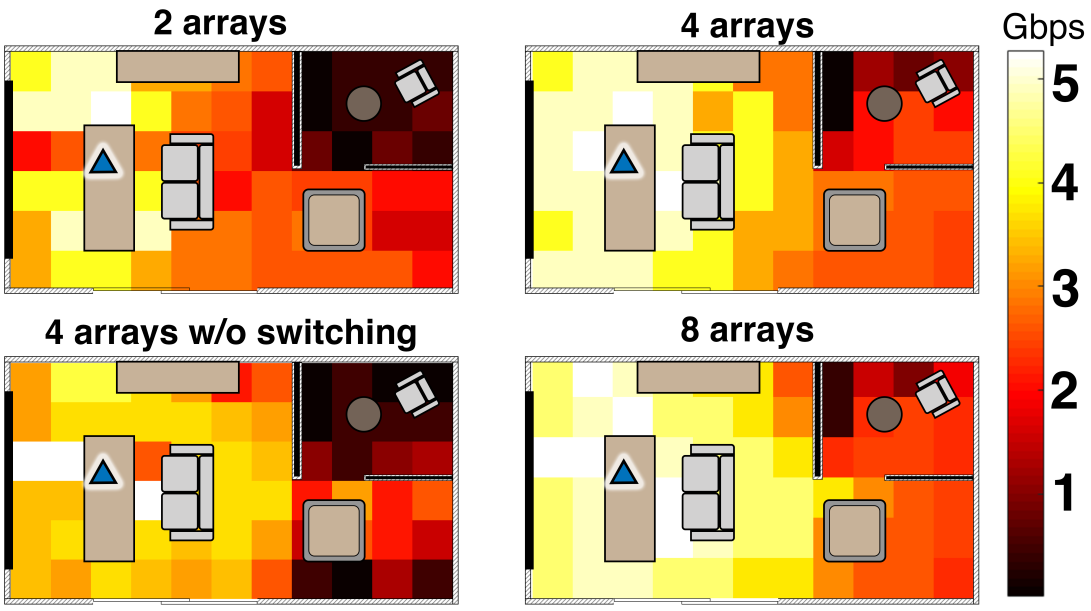


Figure 4.18: Coverage improvement in a room environment.

is due to two reasons. *First*, the former turns on all arrays, and wastes transmit power on arrays that may not provide any multipath diversity. Interestingly, we found that for 4-Array and 8-Array cases running X-Array, only 2 arrays are activated most of the time, implying it can intelligently select the best arrays rather than turning on all. *Second*, the phases between panels in 4-array w/o switching case are randomly chosen and not coherently aligned most of the time. Consequently, the throughput performance is harmed and becomes unstable, implying the importance of the co-phasing design.

4.6 Discussion

APA represents a relatively new phased array architecture to establish high-performance mmWave networks. Our X-Array system has addressed several major challenges in APA,

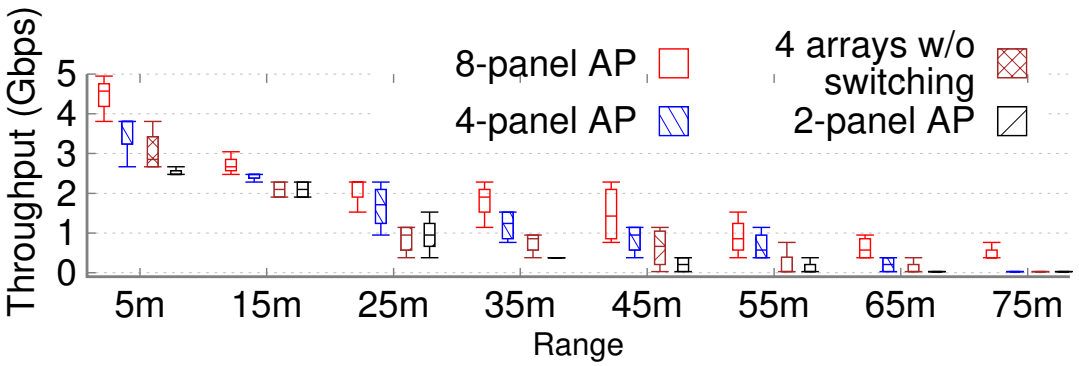


Figure 4.19: Multi-array outdoor range improvement.

but many other design choices exist, which we discuss below.

Extension to multi-RF-chain mmWave MIMO. Emerging mmWave network standards such as 802.11ay and 5G NR support mmWave MIMO, *i.e.*, multiple RF chains each connecting to one phased array, sending multiple streams of data simultaneously to a single user (SU-MIMO multiplexing mode) or multiple users (MU-MIMO). As mentioned in Sec. 4.3, extension to multiple users in single-RF chain device can be realized using the built-in MAC protocol in 802.11ad like CSMA/TDMA based scheduling. Alternatively, mmWave MIMO can send the same stream of data across multiple RF chains to a single user to improve its SNR (SU-MIMO diversity mode). Although APA has a single RF chain, it can be considered as a special case of SU-MIMO diversity. The AoD estimation, array/beam selection and outage handling mechanisms in X-Array can thus be directly applied to facilitate SU-MIMO. Its co-phasing implementation has been constrained by the 802.11ad hardware (2-bit phase resolution), yet the dynamic co-phasing formulation is general enough for future 802.11ay devices with phase weights. Even for the mmWave

multi-user MIMO (MU-MIMO) case, X-Array's three key mechanisms can serve as essential facilitating functions. Practical mmWave MU-MIMO protocols need to separate the analog beamforming and digital beamforming in two steps [145]. The former still heavily relies on the angular estimation and beam selection to narrow down the beam search space and then identify the best beam on each array for each user. These challenges are similar in nature to APA and can still benefit from the basic design components of X-Array. The detailed design that integrates X-Array in mmWave MIMO is left for our future work.

Dealing with more versatile client devices. Our current X-Array design assumes a simple single-array mobile client device. But future mmWave mobile devices may encompass APA to overcome users' hand/body blockage. Accordingly, the array selection mechanism needs to be updated to coordinate the AP and client simultaneously. In addition, X-Array derives its decisions mainly based on the 802.11ad SLS, when the client turns to quasi-omni mode. Ideally, the client can further select its receiving beam after the AP's array/beam are selected. This function is not implementable on our current 802.11ad device, but may be explored when an APA software radio becomes available.

Impact on higher layers Dynamic co-phasing of X-Array prevents most of the sudden drops of link quality (indicated by link RSS). Such variation will severely affect the effective throughput on the higher layer protocols and applications that are sensitive to instantaneous bandwidth estimation (*e.g.* TCP and DASH) [147, 148]. Such amplified impacts on higher layers are well studied and beyond the scope of our current work.

Multi-APA interference and spatial reuse. Our present work focuses on optimizing a single-cell mmWave network with a single AP. When multiple clients co-exist,

the decision of each can be made independently, and the overhead will not increase in a noticeable way, because all of them share the SLS broadcast beacons from the AP. When multiple APs and clients coexist, X-Array may not sacrifice spatial reuse much since it tries to direct power towards the dominant AoA. Proper interference management schemes (*e.g.*, [149]) may still be needed, but are beyond the scope of this work.

4.7 Related Work

In order to overcome the two main obstacles in realizing robust mmWave networking, *i.e.* mobility and blockage, recent research has explored efficient beam management algorithms, along with new network architectures.

By reducing beam scanning overhead, the transmitter and receiver can quickly realign their beams, thus becoming resilient under channel dynamics. AgileLink [118] makes use of multi-arm beams and Hash function to identify the signal power along all spatial directions, and selects the beam along the strongest direction. BeamSpy [117] learns the correlation between beams offline, and prunes the beam search space to efficiently recover from blockage. UbiG [150] introduces an asymptotically efficient beam alignment algorithm that uses a few probings to estimate the best beam. Most of such algorithms, along with many compressive sensing and statistical estimation algorithms[151, 152, 153], rely on the CSI as input, which requires non-trivial on-board memory space [133] and is unavailable on typical commodity mmWave radios. ACO [141] acquires CSI indirectly by measuring the RSS corresponding to different phase shift values of different groups antenna

elements. Based on the CSI, it [141] estimates AoD according to [146] and form beams accordingly. However, the CSI acquisition process itself takes non-trivial overhead.

To overcome the FoV limitations of phased-arrays, Pia [124] leverages multiple cooperative APs, and switches to the appropriate AP whenever one is blocked. It uses motion and location sensors on mobile mmWave devices to overcome user mobility and orientation changes. Bouncenet [149] further addresses the spatial reuse when multiple APs and clients coexist. EMi [154] first reconstructs the reflection environment using mmWave sensing, and then intelligently places the APs to improve long-term network robustness under random blockage and mobility. Beam-forecast and miDroid [119, 155] also leverages the environment information, and matches the measured CSI with ray-tracing simulated CSI in order to guide the beam selection. Yang *et al.* [156] leverages the mobile sensor data to adapt the beamwidth by choosing the beam from a multi-level codebook. Listeer [157] makes use of luminaries information from lighting infrastructure to help with maintaining beam alignment and tracking mobility. Notably, none of the above work addresses the challenges related to the APA architecture (Sec. 4.2.2).

In emerging mmWave SU/MU-MIMO standards such as 802.11ay, the AP first performs legacy beam scanning and let clients report a set of potential beams for each phased array. Then the AP collects CSI feedback of the clients' selected beams and further performs digital precoding to realize hybrid beamforming. Recent work designed algorithms to group the clients [158] or estimate the mmWave MIMO channel [159, 160], assuming detailed CSI feedback is available. As discussed in Sec. 4.6, X-Array addresses a different set of problems, although its design components can be transferable to single

user mmWave MIMO.

4.8 Conclusion

We have explored APA as a new paradigm to simultaneously improve the mmWave coverage and link quality. Our X-Array solution framework builds on the 802.11ad standard and runs directly on commodity devices. Our experiments have verified X-Array's advantages in terms of efficiency, coverage, and ability to rapidly recover from link outage. We believe X-Array marks an important step in making mmWave networks more resilient in dynamic and mobile scenarios.

Chapter 4 contains material from "X-array: Approximating Omnidirectional Millimeter-Wave Coverage Using an Array of Phased Arrays" by Song Wang, Jingqi Huang, Xinyu Zhang, Hyoil Kim, and Sujit Dey, which appears in the ACM International Conference On Mobile Computing And Networking, 2020. The dissertation author was the primary investigator and author of this paper.

Chapter 5

Summary and Future Work

The escalating scale of ML models, coupled with the growing demand for mobile ML applications, necessitates innovative solutions in mobile ML. The traditional centralized paradigms, such as cloud and MEC based ML, fall short of performance due to their inability to adapt to the dynamics of mobile networks. Split ML, as the distributed mobile ML paradigm, links the computing nodes within the mobile network, amalgamating them into a significantly more potent and intelligent collective mind. Through the exploration of Split ML in this dissertation, we believe that we are forging a path towards the delivery of real-time Artificial General Intelligence (AGI) to every mobile user.

5.1 Dissertation Summary

In this dissertation, we thoroughly explore the design aspects of Split ML, identifying key challenges and deploying a variety of techniques—such as a unique split algorithm,

innovative communication protocols, and hardware designs—to implement an effective Split ML system on mobile edge networks.

We first introduce HiveMind, the first multi-split ML system designed for 5G cellular networks. HiveMind simplifies the complex multi-split problem into a mini-cost graph search, optimizing the distributed algorithm to drastically reduce signaling overhead. Thanks to its low overhead, HiveMind facilitates real-time optimal split decisions across multiple computing nodes and adapts to instantaneous network dynamics. It also integrates a multi-objective mechanism that accommodates diverse objectives for a single ML task. HiveMind is compatible with a wide variety of ML frameworks, including non-linear models like Recurrent Neural Network, Federated Learning, and Multi-agent Reinforcement Learning. Evaluation of HiveMind using 5G MEC network simulators with realistic traffic patterns and real-life MEC computation/communication profiles demonstrates that it outperforms current split ML designs in efficiency.

We further characterizes the error tolerance capability of split ML intermediate data communications. From these insights, we introduce NeuroMessenger, a lightweight mechanism integrated into the cellular network stack to exploit ML data's error tolerance and reduce communication overhead. NeuroMessenger simplifies development and deployment as it does not require per-model profiling and remains transparent to the application layer. Experiments on a 5G simulation framework demonstrate that NeuroMessenger decreases end-to-end latency by up to 99% while maintaining low accuracy loss under various link conditions.

Finally, we propose X-Array, an innovative mmWave radio architecture designed

to accommodate the bandwidth needs of split ML. X-Array simultaneously selects arrays and beams, applying a dynamic co-phasing mechanism to ensure signal enhancement across different arrays. It also features a link recovery mechanism to identify alternative arrays/beams for efficient link recovery from outage. Implemented on a commercial 802.11ad APA radio, our experiments show that X-Array achieves near omni-directional coverage and maintains high performance despite link dynamics.

5.2 Future Work

5.2.1 Limitations of Existing Works

This dissertation presents an initial exploration of Split ML systems, based on certain assumptions that may guide future extensions.

Firstly, it assumes that each server in the split ML system serves a single user, eliminating the need to schedule computational resources. However, in practical deployments, computational resources should be scheduled with a clear understanding of the split options to maximize system efficiency.

Secondly, the dissertation posits that the runtime for each ML inference and training pass is significantly shorter than the interval between calls to the split ML service. This would mean that computation on one node wouldn't be hampered by the preceding node. Yet, in real-world scenarios, communication latencies can fluctuate greatly. Therefore, a queuing mechanism might be necessary to handle potential bottlenecks in split workloads.

Lastly, the study is predicated on the use of 5G cellular networks, known for their

expansive bandwidth. However, when dealing with IoT devices equipped with limited computational capabilities and low-data-rate radios, such as ZigBee, the balance between computation and communication expenses becomes more acute. This necessitates innovative design approaches.

5.2.2 Orchestration of Multi-tenant Split ML Deployment on MEC

The work presented in this dissertation considers only single-user split ML service usage. However, in practical MEC deployments, a split ML service must cater to multiple users sharing communication and computational resources allocated for the split ML service. Consequently, effective resource scheduling is necessary to ensure optimal overall split ML performance while maintaining fairness.

The complexity of this scheduling challenge surpasses that of traditional scheduling in distributed systems, due to two main factors. Firstly, resource scheduling complexity increases due to split options. As demonstrated in Chapter 2, an ML model's optimal split option is influenced by computation and communication resources, which in turn determine a user's required resources. Therefore, scheduling and split option decisions must be jointly optimized, increasing problem complexity.

Secondly, scheduling decisions must account for both computation and communication resources. Unlike traditional scheduling designs operating on stable data center links, split ML scheduling operates on more dynamic, unstable mobile links. This necessitates

more frequent scheduling decisions to accommodate these dynamics. To make informed decisions, the scheduler requires mobile link information, such as link quality and block error rates, from the integrated Radio Network Information Service (RNIS) within the MEC system [15]. However, frequent RNIS calls burden the MEC control plane and undermine the overall efficiency of the MEC system. As such, a scheduling system capable of making real-time joint scheduling and split decisions with limited access to network telemetry information is required.

5.2.3 UWB-based Split Spiking Neural Networks

Despite split ML's success in alleviating computational pressure on mobile devices, growing ML model sizes continue to strain data centers and MEC sites. Specifically, ML services' energy consumption, projected to reach 0.4 TWh by 2021 [161], grows commensurate with model size, presenting significant environmental challenges. To combat the energy-intensive nature of traditional neural networks, Spiking Neural Networks (SNN) have been proposed. Mimicking the behavior of neural cells, SNNs employ a series of spikes to encode information, eschewing digitized numbers. Executed on specialized neuromorphic chips [162], SNNs significantly outperform traditional GPUs or TPUs in terms of power efficiency. For identical models, an SNN consumes only 1/17 the energy of a traditional neural network, achieving the same results [163].

Like traditional neural networks, SNNs maintain a layered structure, making them compatible with split ML systems. To optimize energy efficiency in split SNNs, the communication system's energy consumption must be equivalent to that of SNN computation.

In split SNNs, intermediate data is spike series, which must be digitized before digital communication system integration—increasing energy consumption. Moreover, digital communication systems demand 2 to 5× more energy than SNN computation. Instead, pulse-based systems, such as UWB, offer a viable alternative. By representing SNN spikes as pulses and forgoing complex DAC and clocking systems, UWB offers considerable energy advantages over digital systems. Consequently, we posit that a UWB-based split SNN communication system represents a significant step towards realizing our vision of low-power mobile intelligence.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[3] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4401–4410.

[4] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and improving the image quality of stylegan," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 8110–8119.

[5] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.

[6] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[7] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, "Segment anything," *arXiv:2304.02643*, 2023.

[8] Counterpoint Research. (2020) Average smartphone NAND flash capacity crossed 100 GB in 2020. [Online]. Available: https://www.counterpointresearch.com/average-smartphone-nand-flash-capacity-crossed-100gb-2020/

[9] A. W. Services, "Artificial intelligence – ai and machine learning," 2023. [Online]. Available: https://aws.amazon.com/ai/

[10] Microsoft, "Azure machine learning," 2023. [Online]. Available: https://azure.microsoft.com/en-us/products/machine-learning/

[11] 3GPP, "5g system (5gs); study on traffic characteristics and performance requirements for ai/ml model transfer," TR 22.874 V0.0.0, 2020.

[12] 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, pp. 1738–1762, 2019.

[13] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[14] C. G. C. Index, "Forecast and methodology, 2016–2021 white paper," *Updated: February*, vol. 1, p. 2018, 2018.

[15] S. Kekki, W. Featherstone, Y. Fang, P. Kuure, A. Li, A. Ranjan, D. Purkayastha, F. Jiangping, D. Frydman, G. Verin *et al.*, "Mec in 5g networks," *ETSI white paper*, vol. 28, pp. 1–28, 2018.

[16] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[17] S. Cao, L. Ma, W. Xiao, C. Zhang, Y. Liu, L. Zhang, L. Nie, and Z. Yang, "Seernet: Predicting convolutional neural network feature-map sparsity through low-bit quantization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 216–11 225.

[18] F. Giust, G. Verin, K. Antevski, J. Chou, Y. Fang, W. Featherstone, F. Fontes, D. Frydman, A. Li, A. Manzalini *et al.*, "Mec deployments in 4g and evolution towards 5g," *ETSI White Paper*, vol. 24, pp. 1–24, 2018.

[19] L. Google, "Ml kit — google developers," https://developers.google.com/ml-kit, 2020.

[20] A. Inc., "Core ml — apple developer documentation," https://developer.apple.com/documentation/coreml, 2020.

[21] austin kodra, "Awesome-mobile-machine-learning," https://github.com/fritzlabs/Awesome-Mobile-Machine-Learning, 2020.

[22] M. Xu, J. Liu, Y. Liu, F. X. Lin, Y. Liu, and X. Liu, "A first look at deep learning apps on smartphones," in *The World Wide Web Conference*, 2019, pp. 2125–2136.

[23] ITU, "Y.3172 : Architectural framework for machine learning in future networks including IMT-2020," Jun. 2019.

[24] S.-C. Lin, I. F. Akyildiz, P. Wang, and M. Luo, "QoS-Aware Adaptive Routing in Multi-layer Hierarchical Software Defined Networks: A Reinforcement Learning Approach," in *IEEE International Conference on Services Computing 2016*, 2016.

[25] Z. Lin and M. van der Schaar, "Autonomic and Distributed Joint Routing and Power Control for Delay-Sensitive Applications in Multi-Hop Wireless Networks," *IEEE Trans. on Wireless Communications*, 2011.

[26] K.-L. A. Yau, J. Qadir, C. Wu, M. A. Imran, and M. H. Ling, "Cognition-Inspired 5G Cellular Networks: A Review and the Road Ahead," *IEEE Access*, vol. 6, 2018.

[27] Y. Xiao, G. Shi, Y. Li, W. Saad, and H. V. Poor, "Toward self-learning edge intelligence in 6g," *IEEE Communications Magazine*, vol. 58, no. 12, pp. 34–40, 2020.

[28] I. Stoica, D. Song, R. A. Popa, D. Patterson, M. W. Mahoney, R. Katz, A. D. Joseph, M. Jordan, J. M. Hellerstein, J. E. Gonzalez *et al.*, "A berkeley view of systems challenges for ai," *arXiv preprint arXiv:1712.05855*, 2017.

[29] E. Li, Z. Zhou, and X. Chen, "Edge intelligence: On-demand deep learning model co-inference with device-edge synergy," in *Proceedings of the 2018 Workshop on Mobile Edge Communications*, 2018, pp. 31–36.

[30] 3GPP, "Nr; study on integrated access and backhaul," TR 38.874 V16.0.0, 2019.

[31] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, "More effective distributed ml via a stale synchronous parallel parameter server," in *Advances in neural information processing systems*, 2013, pp. 1223–1231.

[32] A. Agarwal, O. Chapelle, M. Dudík, and J. Langford, "A reliable effective terascale linear learning system," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1111–1133, 2014.

[33] I. Foster and A. Iamnitchi, "On death, taxes, and the convergence of peer-to-peer and grid computing," in *International Workshop on Peer-To-Peer Systems*. Springer, 2003, pp. 118–128.

[34] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 2, pp. 1–19, 2019.

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

[36] C. Niu, F. Wu, S. Tang, L. Hua, R. Jia, C. Lv, Z. Wu, and G. Chen, "Billion-scale federated learning on mobile clients: a submodel design with tunable privacy," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, 2020, pp. 1–14.

[37] Q. Chen, Z. Zheng, C. Hu, D. Wang, and F. Liu, "Data-driven task allocation for multi-task transfer learning on the edge," in *2019 IEEE 39th International Conference on Distributed Computing Systems*. IEEE, 2019, pp. 1040–1050.

[38] 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*. Proceedings of Machine Learning Research, 2017, pp. 1273–1282.

[39] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp. 1310–1321.

[40] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive dnn surgery for inference acceleration on the edge," in *2019 IEEE International Conference on Computer Communications*. IEEE, 2019, pp. 1423–1431.

[41] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "Pipedream: generalized pipeline parallelism for dnn training," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 1–15.

[42] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition*. IEEE, 2016, pp. 2464–2469.

[43] ——, "Distributed deep neural networks over the cloud, the edge and end devices," in *2017 IEEE 37th International Conference on Distributed Computing Systems*. IEEE, 2017, pp. 328–339.

[44] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama, "Adaptive neural networks for efficient inference," *arXiv preprint arXiv:1702.07811*, 2017.

[45] C. Lo, Y.-Y. Su, C.-Y. Lee, and S.-C. Chang, "A dynamic deep neural network design for efficient workload allocation in edge computing," in *2017 IEEE International Conference on Computer Design*. IEEE, 2017, pp. 273–280.

[46] S. Leroux, S. Bohez, E. De Coninck, T. Verbelen, B. Vankeirsbilck, P. Simoens, and B. Dhoedt, "The cascading neural network: building the internet of smart things," *Knowledge and Information Systems*, vol. 52, no. 3, pp. 791–814, 2017.

[47] L. Li, K. Ota, and M. Dong, "Deep learning for smart industry: Efficient manufacture inspection system with fog computing," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4665–4673, 2018.

[48] X. Xie and K.-H. Kim, "Source compression with bounded dnn perception loss for iot edge computer vision," in *The 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1–16.

[49] H. Li, C. Hu, J. Jiang, Z. Wang, Y. Wen, and W. Zhu, "Jalad: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution," in *2018 IEEE 24th International Conference on Parallel and Distributed Systems*. IEEE, 2018, pp. 671–678.

[50] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *arXiv preprint arXiv:1712.01887*, 2017.

[51] S. U. Stich, J.-B. Cordonnier, and M. Jaggi, "Sparsified sgd with memory," in *Advances in Neural Information Processing Systems*, 2018, pp. 4447–4458.

[52] H. Tang, S. Gan, C. Zhang, T. Zhang, and J. Liu, "Communication compression for decentralized training," in *Advances in Neural Information Processing Systems*, 2018, pp. 7652–7662.

[53] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[54] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[55] S. Yao, J. Li, D. Liu, T. Wang, S. Liu, H. Shao, and T. Abdelzaher, "Deep compressive offloading: speeding up neural network inference by trading edge computation for network latency," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 2020, pp. 476–488.

[56] 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*, 2016, pp. 770–778.

[57] P. A. Humblet, "Another adaptive distributed shortest path algorithm," *IEEE transactions on communications*, vol. 39, no. 6, pp. 995–1003, 1991.

[58] M. S. Corson and A. Ephremides, "A distributed routing algorithm for mobile wireless networks," *Wireless networks*, vol. 1, no. 1, pp. 61–81, 1995.

[59] D. Xu, A. Zhou, X. Zhang, G. Wang, X. Liu, C. An, Y. Shi, L. Liu, and H. Ma, "Understanding operational 5g: A first measurement study on its coverage, performance and energy consumption," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 479–494.

[60] J. Ding, X. Liu, Y. Li, D. Wu, D. Jin, and S. Chen, "Measurement-driven capability modeling for mobile network in large-scale urban environment," in *2016 IEEE 13th International Conference on Mobile Ad Hoc and Sensor Systems*. IEEE, 2016, pp. 92–100.

[61] C. Yue, R. Jin, K. Suh, Y. Qin, B. Wang, and W. Wei, "Linkforecast: cellular link bandwidth prediction in lte networks," *IEEE Transactions on Mobile Computing*, vol. 17, no. 7, pp. 1582–1594, 2017.

[62] ETSI, "Multi-access edge computing (mec); support for network slicing," ETSI GR MEC 024 V2.1.1, 2019.

[63] P. Manyem and J. Ugon, "Computational complexity, np completeness and optimization duality: A survey," *Electron. Colloquium Comput. Complex.*, vol. 19, p. 9, 2012.

[64] sgrvinod, "a-pytorch-tutorial-to-sequence-labeling," 2020. [Online]. Available: https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Sequence-Labeling

[65] H. Pan, Z. Li, J. Dong, Z. Cao, T. Lan, D. Zhang, G. Tyson, and G. Xie, "Dissecting the communication latency in distributed deep sparse learning," in *Proceedings of the ACM Internet Measurement Conference*, 2020, pp. 528–534.

[66] 3GPP, "Nr; physical channels and modulation," TR 38.214 V16.4.0, 2021.

[67] Federal Communications Commission, "FCC 15-138A1," 2015.

[68] 3GPP, "Nr; physical layer procedures for data," TR 38.214 V16.4.0, 2021.

[69] R. Kwan, C. Leung, and J. Zhang, "Proportional fair multiuser scheduling in lte," *IEEE Signal Processing Letters*, vol. 16, no. 6, pp. 461–464, 2009.

[70] R. R. Chakib Belgaid, Arthur d'Azémar, "PyJoules PyPi," https://pypi.org/project/pyJoules/, 2020.

[71] F. Malandrino, C.-F. Chiasserini, and S. Kirkpatrick, "Cellular network traces towards 5g: Usage, analysis and generation," *IEEE Transactions on Mobile Computing*, vol. 17, no. 3, pp. 529–542, 2017.

[72] T. Rashid, M. Samvelyan, C. S. De Witt, G. Farquhar, J. Foerster, and S. Whiteson, "Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning," *arXiv preprint arXiv:1803.11485*, 2018.

[73] E. Peltonen, M. Bennis, M. Capobianco, M. Debbah, A. Ding, F. Gil-Castiñeira, M. Jurmu, T. Karvonen, M. Kelanti, A. Kliks *et al.*, "6g white paper on edge intelligence," *arXiv preprint arXiv:2004.14850*, 2020.

[74] 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, pp. 2322–2358, 2017.

[75] F. Boccardi, R. W. Heath, A. Lozano, T. L. Marzetta, and P. Popovski, "Five disruptive technology directions for 5g," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 74–80, 2014.

[76] M. G. Arivazhagan, V. Aggarwal, A. K. Singh, and S. Choudhary, "Federated learning with personalization layers," *CoRR*, vol. abs/1912.00818, 2019.

[77] H. Wang, M. Yurochkin, Y. Sun, D. S. Papailiopoulos, and Y. Khazaeni, "Federated learning with matched averaging," *CoRR*, vol. abs/2002.06440, 2020.

[78] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–7.

[79] N. Yoshida, T. Nishio, M. Morikura, K. Yamamoto, and R. Yonetani, "Hybrid-fl for wireless networks: Cooperative learning mechanism using non-iid data," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–7.

[80] T. T. Anh, N. C. Luong, D. Niyato, D. I. Kim, and L.-C. Wang, "Efficient training management for mobile crowd-machine learning: A deep reinforcement learning approach," *IEEE Wireless Communications Letters*, vol. 8, no. 5, pp. 1345–1348, 2019.

[81] J. Kang, Z. Xiong, D. Niyato, S. Xie, and J. Zhang, "Incentive mechanism for reliable federated learning: A joint optimization approach to combining reputation and contract theory," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10 700–10 714, 2019.

[82] J. Kang, Z. Xiong, D. Niyato, H. Yu, Y.-C. Liang, and D. I. Kim, "Incentive design for efficient federated learning in mobile networks: A contract theory approach," in *2019 IEEE VTS Asia Pacific Wireless Communications Symposium (APWCS)*, 2019, pp. 1–5.

[83] P. Hu, J. Im, Z. Asgar, and S. Katti, "Starfish: resilient image compression for aiot cameras," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 2020, pp. 395–408.

[84] M. Hamza, A. Lipovac, and V. Lipovac, "Residual block error rate prediction for ir harq protocol," *Tehnički vjesnik*, vol. 27, no. 4, pp. 1071–1076, 2020.

[85] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[86] 3GPP, "Nr; physical layer; general description," TS 38.201 V16.0.0, 2020.

[87] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. PMLR, 20–22 Apr 2017, pp. 1273–1282.

[88] C.-Y. Hsu, A. Ortega, and M. Khansari, "Rate control for robust video transmission over burst-error wireless channels," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 5, pp. 756–773, 1999.

[89] S. Lin and D. J. Costello, *Error control coding*. Prentice hall, 2001, vol. 2, no. 4.

[90] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE transactions on information theory*, vol. 52, no. 6, pp. 2508–2530, 2006.

[91] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[92] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen, J. H. Engel, L. Fan, C. Fougner, T. Han, A. Y. Hannun, B. Jun, P. LeGresley, L. Lin, S. Narang, A. Y. Ng, S. Ozair, R. Prenger, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, Y. Wang, Z. Wang, C. Wang, B. Xiao, D. Yogatama, J. Zhan, and Z. Zhu, "Deep speech 2: End-to-end speech recognition in english and mandarin," *CoRR*, vol. abs/1512.02595, 2015.

[93] S. Yao, J. Li, D. Liu, T. Wang, S. Liu, H. Shao, and T. Abdelzaher, "Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, ser. SenSys '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 476–488. [Online]. Available: https://doi.org/10.1145/3384419.3430898

[94] S. Jakubczak and D. Katabi, "A cross-layer design for scalable mobile video," in *Proceedings of the 17th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 289–300.

[95] ETSI, "5g;management and orchestration; architecture framework," ETSI TS 128 5ss V16.4.0, 2020.

[96] M. Kanj, V. Savaux, and M. Le Guen, "A tutorial on nb-iot physical layer design," *IEEE Communications Surveys & Tutorials*, 2020.

[97] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[98] PyTorch, "torch.sparse — pytorch 1.9.0 documentation," 2020. [Online]. Available: https://pytorch.org/docs/stable/sparse.html

[99] MathWorks, "Nr pusch throughput," 2020. [Online]. Available: https://www.mathworks.com/help/5g/ug/nr-pusch-throughput.html

[100] 3GPP, "Study on channel model for frequency spectrum above 6 ghz," TR 38.900 V15.0.0, 2018.

[101] Federal Communications Commission, "FCC Promotes Higher Frequency Spectrum for Future Wireless Technology," https://apps.fcc.gov, 2015.

[102] T. S. Rappaport, S. Sun, R. Mayzus, H. Zhao, Y. Azar, K. Wang, G. N. Wong, J. K. Schulz, M. Samimi, and F. Gutierrez, "Millimeter wave mobile communications for 5g cellular: It will work!" *IEEE access*, vol. 1, pp. 335–349, 2013.

[103] The White House, "Unlocking the Promise of Broadband for All Americans." https://www.whitehouse.gov/blog/2016/07/15/unlocking- promise-broadband-generate-gains-all-americans., 2016.

[104] Cisco Systems, Inc, "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016-2021 White Paper," https://www.cisco.com/c/en/us/solutions/collateral/service- provider/visual-networking-index-vni/mobile-white-paper-c11- 520862.html, 2016.

[105] O. Abari, D. Bharadia, A. Duffield, and D. Katabi, "Cutting the cord in virtual reality," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM, 2016, pp. 162–168.

[106] K. Guan, D. He, A. Hrovat, B. Ai, Z. Zhong, and T. Kürner, "Challenges and chances for smart rail mobility at mmwave and thz bands from the channels viewpoint," in *2017 15th International Conference on ITS Telecommunications (ITST)*. IEEE, 2017, pp. 1–5.

[107] S. E. Elayoubi, M. Fallgren, P. Spapis, G. Zimmermann, D. Martín-Sacristán, C. Yang, S. Jeux, P. Agyapong, L. Campoy, Y. Qi, and S. Singh, "5G Service Requirements and Operational Use Cases: Analysis and METIS II Vision," in *European Conference on Networks and Communications (EuCNC)*, 2016.

[108] M. Tercero, P. von Wrycza, A. Amah, J. Widmer, M. Fresia, V. Frascolla, J. Lorca, T. Svensson, M. Hamon, S. Destouet Roblot, A. Vijay, M. Peter, V. Sgardoni, M. Hunukumbure, J. Luo, and N. Vucic, "5G Systems: The mmMAGIC Project Perspective on Use Cases and Challenges Between 6−100 GHz," in *IEEE Wireless Communications and Networking Conference*, 2016.

[109] C. Tranoris, S. Denazis, L. Guardalben, J. Pereira, and S. Sargento, "Enabling Cyber-Physical Systems for 5G Networking: A Case Study on the Automotive Vertical Domain," in *IEEE/ACM International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS)*, 2018.

[110] M. Boers, B. Afshar, I. Vassiliou, S. Sarkar, S. T. Nicolson, E. Adabi, B. G. Perumana, T. Chalvatzis, S. Kavvadias, P. Sen, W. L. Chan, A. H. Yu, A. Parsa, M. Nariman, S. Yoon, A. G. Besoli, C. A. Kyriazidou, G. Zochios, J. A. Castaneda, T. Sowlati, M. Rofougaran, and A. Rofougaran, "A 16TX/16RX 60 GHz 802.11ad Chipset With Single Coaxial Interface and Polarization Diversity," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 12, 2014.

[111] R. J. Mailloux, *Phased array antenna handbook*. Artech house, 2017.

[112] P. Hannan, "The element-gain paradox for a phased-array antenna," *IEEE Transactions on Antennas and Propagation*, vol. 12, no. 4, pp. 423–433, 1964.

[113] C. A. Balanis, *Antenna theory: analysis and design*. John wiley & sons, 2016.

[114] B. Li, Z. Zhou, W. Zou, X. Sun, and G. Du, "On the efficient beam-forming training for 60ghz wireless personal area networks," *IEEE Transactions on Wireless Communications*, vol. 12, no. 2, pp. 504–515, 2012.

[115] S. Sur, V. Venkateswaran, X. Zhang, and P. Ramanathan, "60 ghz indoor networking through flexible beams: A link-level profiling," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 1. ACM, 2015, pp. 71–84.

[116] A. Patra, L. Simić, and P. Mähönen, "Smart mm-wave beam steering algorithm for fast link re-establishment under node mobility in 60 ghz indoor wlans," in *Proceedings of the 13th ACM International Symposium on Mobility Management and Wireless Access*. ACM, 2015, pp. 53–62.

[117] S. Sur, X. Zhang, P. Ramanathan, and R. Chandra, "BeamSpy: Enabling Robust 60 GHz Links Under Blockage," in *Proceedings of Usenix Conference on Networked Systems Design and Implementation (NSDI)*, 2016.

[118] O. Abari, H. Hassanieh, M. Rodriguez, and D. Katabi, "Millimeter wave communications: From point-to-point links to agile network connections," ACM, pp. 169–175, 2016.

[119] A. Zhou, X. Zhang, and H. Ma, "Beam-forecast: Facilitating mobile 60 ghz networks via model-driven beam steering," in *IEEE Conference on Computer Communications*. IEEE, 2017.

[120] J. Palacios, D. De Donno, and J. Widmer, "Tracking mm-wave channel dynamics: Fast beam training strategies under mobility," in *IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[121] T. Sowlati, S. Sarkar, B. G. Perumana, W. L. Chan, A. Papio Toda, B. Afshar, M. Boers, D. Shin, T. R. Mercer, W. Chen, A. Grau Besoli, S. Yoon, S. Kyriazidou, P. Yang, V. Aggarwal, N. Vakilian, D. Rozenblit, M. Kahrizi, J. Zhang, A. Wang, P. Sen, D. Murphy, A. Sajjadi, A. Mehrabani, E. Kornaros, K. Low, K. Kimura, V. Roussel, H. Xie, and V. Kodavati, "A 60-GHz 144-Element Phased-Array Transceiver for Backhaul Application," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 12, 2018.

[122] T. Sowlati, S. Sarkar, B. Perumana, W. L. Chan, B. Afshar, M. Boers, D. Shin, T. Mercer, W. Chen, A. P. Toda, A. G. Besoli, S. Yoon, S. Kyriazidou, P. Yang, V. Aggarwal, N. Vakilian, D. Rozenblit, M. Kahrizi, J. Zhang, A. Wang, P. Sen, D. Murphy, M. Mikhemar, A. Sajjadi, A. Mehrabani, B. Ibrahim, B. Pan, K. Juan, S. Xu, C. Guan, G. Geshvindman, K. Low, N. Kocaman, H. Eberhart, K. Kimura, I. Elgorriaga, V. Roussel, H. Xie, L. Shi, and V. Kodavati, "A 60GHz 144-element phased-array transceiver with 51dBm maximum EIRP and ±60° beam steering for backhaul application," in *IEEE International Solid - State Circuits Conference (ISSCC)*, 2018.

[123] K. Ramachandran, N. Prasad, K. Hosoya, K. Maruhashi, and S. Rangarajan, "Adaptive beamforming for 60 ghz radios: Challenges and preliminary solutions," in *Proceedings of the 2010 ACM international workshop on mmWave communications: from circuits to networks*. ACM, 2010, pp. 33–38.

[124] T. Wei and X. Zhang, "Pose information assisted 60 ghz networks: Towards seamless coverage and mobility support," in *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*. ACM, 2017, pp. 42–55.

[125] Microtik, "60 GHz Access Point with 3 Titled Phased Arrays," https://mikrotik.com/product/wap_60gx3_ap/fndtn-specifications, 2017.

[126] "Airfide inc." http://airfidenet.com, 2019.

[127] W. Hong, K. Baek, Y. Lee, Y. Kim, and S. Ko, "Study and Prototyping of Practically Large-Scale mmWave Antenna Systems for 5G Cellular Devices," *IEEE Communications Magazine*, vol. 52, no. 9, 2014.

[128] Y. Huang, Y. Li, H. Ren, J. Lu, and W. Zhang, "Multi-panel mimo in 5g," *IEEE Communications Magazine*, vol. 56, no. 3, pp. 56–61, 2018.

[129] A. Ghosh, "5g new radio (nr): physical layer overview and performance," in *IEEE communication theory workshop*, 2018, pp. 1–38.

[130] E. Onggosanusi, M. S. Rahman, L. Guo, Y. Kwak, H. Noh, Y. Kim, S. Faxer, M. Harrison, M. Frenne, S. Grant *et al.*, "Modular and High-Resolution Channel State Information and Beam Management for 5G New Radio," *IEEE Communications Magazine*, vol. 56, no. 3, 2018.

[131] J. Liu, K. Au, A. Maaref, J. Luo, H. Baligh, H. Tong, A. Chassaigne, and J. Lorca, "Initial access, mobility, and user-centric multi-beam operation in 5g new radio," *IEEE Communications Magazine*, vol. 56, no. 3, pp. 35–41, 2018.

[132] N. Song, P. Wen, H. Sun, and T. Yang, "Multi-panel based hybrid beamforming for multi-user massive mimo," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–6.

[133] S. Sur, I. Pefkianakis, X. Zhang, and K.-H. Kim, "Practical MU-MIMO User Selection on 802.11Ac Commodity Networks," in *Proceedings of ACM MobiCom*, 2016.

[134] D. Carlson, "Breaking Through the Cost Barrier for Phased Arrays," Microwave Journal, 2018.

[135] J. Zhang, X. Zhang, P. Kulkarni, and P. Ramanathan, "OpenMili: A 60 GHz Software Radio Platform With a Reconfigurable Phased-Array Antenna," in *ACM MobiCom*, 2016.

[136] R. Zhao, T. Woodford, T. Wei, K. Qian, and X. Zhang, "M-Cube: A Millimeter-Wave Massive MIMO Software Radio," in *ACM MobiCom*, 2020.

[137] X. Zhang, "Millimeter-Wave V2X Testbed," http://m3.ucsd.edu/, 2020.

[138] IEEE Standards Association, "IEEE Standards 802.11ad-2012: Enhancements for Very High Throughput in the 60 GHz Band," 2012.

[139] Federal Communications Commission, "FCC 13-112," https://apps.fcc.gov/edocs_public/attachmatch/FCC-13-112A1.pdf.

[140] D. De Donno, J. P. Beltrán, D. Giustiniano, and J. Widmer, "Hybrid analog-digital beam training for mmwave systems with low-resolution rf phase shifters," in *2016 IEEE International Conference on Communications Workshops (ICC)*. IEEE, 2016, pp. 700–705.

[141] J. Palacios, D. Steinmetzer, A. Loch, M. Hollick, and J. Widmer, "Adaptive codebook optimization for beam training on off-the-shelf ieee 802.11 ad devices," in *Proceedings of the ACM Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2018.

[142] S. Sur, I. Pefkianakis, X. Zhang, and K.-H. Kim, "WiFi-Assisted 60 GHz Wireless Networks," in *ACM MobiCom*, 2017.

[143] D. Steinmetzer, D. Wegemer, and M. Hollick. (2017) Talon tools: The framework for practical ieee 802.11ad research. [Online]. Available: https://seemoo.de/talon-tools

[144] IEEE 802.11ay Task Group, "Status of Project IEEE 802.11ay," http://www.ieee802.org/11/Reports/tgay_update.html, 2018.

[145] P. Zhou, K. Cheng, X. Han, X. Fang, Y. Fang, R. He, Y. Long, and Y. Liu, "IEEE 802.11ay-based mmWave WLANs: Design Challenges and Solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, 2018.

[146] R. Schmidt, "Multiple emitter location and signal parameter estimation," *IEEE transactions on antennas and propagation*, vol. 34, no. 3, pp. 276–280, 1986.

[147] X. Xie, X. Zhang, and S. Zhu, "Accelerating mobile web loading using cellular link information," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '17. New York, NY, USA: ACM, 2017, pp. 427–439.

[148] X. Xie, X. Zhang, S. Kumar, and L. E. Li, "PiStream: Physical Layer Informed Adaptive Video Streaming over LTE," in *Proc. of ACM MobiCom*, 2015.

[149] S. Jog, J. Wang, J. Guan, T. Moon, H. Hassanieh, and R. R. Choudhury, "Many-to-many beam alignment in millimeter wave networks," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2019.

[150] S. Sur, I. Pefkianakis, X. Zhang, and K.-H. Kim, "Towards scalable and ubiquitous millimeter-wave wireless networks," in *the 24th Annual International Conference on Mobile Computing and Networking*. ACM, 2018, pp. 257–271.

[151] Z. Marzi, D. Ramasamy, and U. Madhow, "Compressive channel estimation and tracking for large arrays in mm-wave picocells," *IEEE Journal of Selected Topics in Signal Processing*, vol. 10, no. 3, pp. 514–527, 2016.

[152] D. Ramasamy, S. Venkateswaran, and U. Madhow, "Compressive adaptation of large steerable arrays." in *ITA*, 2012, pp. 234–239.

[153] ——, "Compressive tracking with 1000-element arrays: A framework for multi-gbps mm wave cellular downlinks," in *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2012, pp. 690–697.

[154] T. Wei, A. Zhou, and X. Zhang, "Facilitating Robust 60 GHz Network Deployment by Sensing Ambient Reflectors," in *Proc. of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.

[155] A. Zhou, S. Xu, S. Wang, J. Huang, S. Yang, T. Wei, X. Zhang, and H. Ma, "Robot Navigation in Radio Beam Space: Leveraging Robotic Intelligence for Seamless MmWave Network Coverage," in *Proceedings of ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2019.

[156] Z. Yang, P. H. Pathak, Y. Zeng, and P. Mohapatra, "Sensor-Assisted Codebook-Based Beamforming for Mobility Management in 60 ghz WLANs," in *IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, 2015.

[157] M. K. Haider, Y. Ghasempour, D. Koutsonikolas, and E. W. Knightly, "Listeer: mmWave Beam Acquisition and Steering by Tracking Indicator LEDs on Wireless APs," in *Proceedings of ACM Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2018.

[158] Y. Ghasempour and E. W. Knightly, "Decoupling beam steering and user selection for scaling multi-user 60 ghz wlans," in *Proceedings of the 18th ACM International Symposium on Mobile Ad Hoc Networking and Computing*. ACM, 2017, p. 10.

[159] Y. Ghasempour, M. K. Haider, C. Cordeiro, D. Koutsonikolas, and E. Knightly, "Multi-Stream Beam-Training for mmWave MIMO Networks," in *Proceedings of ACM MobiCom*, 2018.

[160] A. Alkhateeb, O. El Ayach, G. Leus, and R. W. Heath, "Channel Estimation and Hybrid Precoding for Millimeter Wave Cellular Systems," *IEEE Journal of Selected Topics in Signal Processing*, vol. 8, no. 5, 2014.

[161] D. Patterson, J. Gonzalez, U. Hölzle, Q. Le, C. Liang, L.-M. Munguia, D. Rothchild, D. R. So, M. Texier, and J. Dean, "The carbon footprint of machine learning training will plateau, then shrink," *Computer*, vol. 55, no. 7, pp. 18–28, 2022.

[162] A. R. Young, M. E. Dean, J. S. Plank, and G. S. Rose, "A review of spiking neuromorphic hardware communication systems," *IEEE Access*, vol. 7, pp. 135 606–135 620, 2019.

[163] T.-Y. Liu, A. Mahjoubfar, D. Prusinski, and L. Stevens, "Neuromorphic computing for content-based image retrieval," *Plos one*, vol. 17, no. 4, p. e0264364, 2022.