Building User-Driven Edge Devices

by

Vidushi Goyal

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2023

Doctoral Committee:

Professor Valeria Bertacco, Co-Chair
Associate Professor Reetuparna Das, Co-Chair
Associate Professor Achilleas Anastasopoulos
Associate Professor Jenna Wiens

Vidushi Goyal

vidushi@umich.edu

ORCID iD: 0000-0002-5008-3049

ACKNOWLEDGMENTS

It is my great pleasure to acknowledge all the people who supported me throughout my Ph.D. journey. First and foremost, I would like to thank my advisors, Prof. Reetuparna Das and Prof. Valeria Bertacco, for showing faith in me and constantly guiding me through lows and highs. They always supported my endeavors to explore my research interests. The numerous brainstorming sessions helped me develop critical thinking and grow as a researcher. I learned various research aspects, ranging from precise problem formation to clear presentation of ideas. Thank you again for sharing your knowledge and expertise. I am also grateful for the thorough feedback and advice that I received from my committee members Prof. Jenna Wiens and Prof. Achilleas Anastasopoulos. Thank you for sharing your unique perspectives. It definitely made this dissertation stronger. I would also like to thank Prof. Thomas Wenisch, who laid my strong fundamentals of computer architecture through rigorous coursework and projects. Prof. Todd Austin for sharing his wisdom, experience, and inspiration all along.

The University of Michigan gave me the opportunity to interact with many amazing and like-minded people from across the world. I would genuinely like to thank all the members of ABResearh and Mbits research groups, especially Doowon Lee, Salessawi Yitbarek, Abraham Addisie, Zelalem Aweke, Arun Subramaniyan, Daichi Fujiki, Xiowei Wang, Fitsum Andargie, Lauren Biernacki, Charles Eckert, Andrew McCrabb, Shibo Chen, Tarunesh Verma, and Alireza Khadem. I thoroughly enjoyed my time and conversations with all of you. I would also like to thank CSE Staff and ADA Staff for their support. Especially Magdalena Calvillo for being so prompt and accommodating of my requests. Big thanks to writing specialists Lauren Rudewicz and Annika Pattenaude. Annika, thank you for being so patient.

My Ph.D. journey would have been incomplete without the friendships I forged along the way. Thank you, Nishil Talati, Shruti Nagaraja, and Tarunesh Verma, for your unequivocal support through dark times. I will always cherish our road trips and board game nights. Thank you, Hiwot Kassa, for our extensive dinner sessions. Sravanti Panja for always being a phone call away. Abraham Addisie, Arun Subramaniyam, and Subarno Banerjee for being there to answer my long job search questions. Akshitha Sriraman for being such a great mentor and friend. Thank you for encouraging me, listening to me, and reminding me of my strengths when I felt lost. Thank you, Tamanna Dubey, for your wise life lessons. I learned a lot from you.

Finally, I dedicate my dissertation to my family. My grandfather, Harish Chandra Goyal, for going against the norms and always prioritizing our education. My parents, Vinita Goyal and Subhash Kumar Goyal, for being my rock and trusting my decisions. You let me chase my dreams. Mamma, I wouldn't be able to see the end of this road without you. My sister, Sugandha Goyal, for tolerating all my tantrums and mood swings. For taking over my responsibilities and cutting me some slack. Thank you for always having my back. My brother, Harshit Goyal, for constant reminders about the bigger picture of life. Last but not least, thank you to Vikalp Aggarwal. I am glad I could share this unique experience with you. I am grateful for your patience and understanding. You ensured that everything and everyone was well managed in the back-end while I focused on my goal. Big thanks for being so supportive and bearing the last leg of this journey with me. I also want to express my gratitude to my extended family for their love and kindness throughout this journey.

TABLE OF CONTENTS

CHAPTER

LIST OF FIGURES

FIGURE

LIST OF TABLES

TABLE

LIST OF ACRONYMS

ML Machine Learning

QoS Quality of Service

IoT Internet of Things

SoC System-on-chip

SiP System-in-package

fps Frames per second

VMAF Video Multimethod Assessment Fusion

MLP Multilayer Perceptron

RM Recommendation Model

NoC Network-on-chip

ABSTRACT

Edge devices like smartphones, wearables, and personal assistants have become an integral part of our daily routines. Their ubiquitous and portable nature allows them to operate in any sort of environment. They can be deployed in the wild or at home without requiring a constant power source plugged into them. However, the small form factor and resource-constrained nature of edge devices limit their computation capabilities and, thus, significantly impact the efficiency of tasks performed on edge devices. The application efficiency is directly related to the quality of user experience for the hand-held edge devices; thus, these shortcomings of the edge device impact the user experience as well.

In this dissertation, we develop solutions to address these limitations of edge devices to enhance the performance and energy efficiency of a wide range of user applications processed on edge devices. Our proposed solutions are either low-cost alternatives that can replace expensive silicon or extract extreme efficiencies from already in-use silicon, thus lowering the total cost of ownership of edge devices. Our solutions are driven by three key strategies: 1) cross-component optimizations across the system, 2) leverage user information and preferences in the hardware, and 3) co-design the application and hardware for the edge system. In our first solution, Seesaw, we study user applications for edge devices with tiny microcontrollers and sensors. We propose an end-to-end automated technique to find optimal compute/sensing rates for power-intensive sensors governed by low-power sensors and based on individual users' preferences and inherent sensing capabilities. This elongates battery life with minimal impact on the perceivable user experience.

In our second proposed solution, we customize the machine learning-based image recognition application for each user by creating small and accurate user-specific machine learning models on the resource constraint edge device. This significantly lowers computation demands and memory

footprint without impacting user accuracy. In the next work, Duet, we leverage the user history and profile information to decompose the giant monolithic recommendation model into a separate user and item model. The user model processes user information in a lightweight manner on the local edge device, and its computation is reused by the item model, processing 100s of items at the datacenter. Thus, we offer enhanced privacy along with performance improvement of 6.4x and energy efficiency of 4.6x.

Finally, we present a low-cost and heterogeneous System-in-Package (SiP)-based multi-chiplet interconnect architecture built over the 2.5D stacking interposer technology, which can replace the expensive monolithic system-on-chip (SoC). The proposed architecture exposes high-bandwidth links of the interposer over which we efficiently map popular bandwidth-intensive edge applications to enhance performance and energy efficiency.

CHAPTER 1

Introduction

User devices influence nearly all aspects of twenty-first-century life. These portable gadgets, like smartphones and tablets, dominate various modes/sectors of human interaction, including communication, healthcare, entertainment, and transportation. For example, smartphones connect people worldwide through instant messaging, audio/video calls, and social media platforms; these devices enable users to shop online and attend doctor's appointments via telemedicine. If the ability to do so much from the comfort of our phones has improved our quality of life significantly, it has also made user devices an indispensable part of our day-to-day lives. Furthermore, user devices are not limited to just handheld devices. They come in various shapes and features to cater to increasing user demands. Personal assistant devices like Alexa, smart-cameras like GoPro, fitness trackers like FitBit, and weather monitoring units for precision agriculture like Arable are some of the innovations driven by users' demands.

The number of edge devices, including a wide range of edge devices like smartphones, sensors devices to security cameras, is increasing rapidly. It is forecasted to increase to 6.5B by 2030, an increase of 4B from 2020 [28]. As user devices are becoming more pervasive and smarter, so is their demand for higher computation capabilities. Compared to the start of 2019, the average hours spent by a user per month on mobile apps had increased by 40% [22] during the 2020 pandemic. An increase in the number of mobile apps and user devices has led to the generation of terabytes of raw data that can be processed in two ways: 1) offloading the computation to back-end cloud servers and 2) computing the data locally at the edge device. Offloading computation to the cloud is a simple option as it allows the edge device to act as a lightweight user-interface while computations are carried out on back-end servers, and final results are communicated back to the user device. However, it requires reliable internet connectivity and high bandwidth links to transfer bulky raw data like videos, audios, and images to back-end servers, which adds to end-to-end service latency and consumes much energy. Further, sharing users' personal data with commodity servers is prone to manipulations and cyber-attacks, thus compromising users' privacy. An alternative option is to process data on edge devices. However, this option is bounded by the computing capability of small resource-constrained user devices.

Over the past decade, the performance and computing power of user devices like smartphones has improved drastically. Simultaneously, algorithms/software applications have also evolved dramatically while becoming more complex to enhance user experience. There is still a wide gap between the computing power offered by user devices and the computational demands of emerging user applications like machine learning (ML), recommendation systems, high-definition video recording/playback, etc.

Additionally, the current edge platform landscape is very diverse and heterogeneous. It ranges from giant autonomous vehicle (AV) systems with compute-intensive sensors and server-like capacity; handheld smartphones containing multi-core ARM architectures and application-specific IPs connected through high bandwidth links; to tiny IoT devices that host small sensors powered by tiny microcontrollers. Moreover, the mobile phone market in itself is very diverse. Facebook [221] has reported that 72% of its users still use 6-year-old Systems-on-Chips. Only one-fourth of its users own phones with CPUs designed in 2015 and later. This broad spectrum in computing capacity shows that in the universe of applications for edge devices, there is no ”one-size-fits-all” solution. Edge device solutions must be tailored for the application and the underlying architecture. However, irrespective of the device, all innovations to improve edge devices are driven by the common goal of offering an enhanced user experience.

Over the past few years, researchers have striven to achieve this goal by applying three approaches: developing software optimizations for applications, building application-specific hardware accelerators, and transitioning the hardware to sophisticated but very expensive silicon technology nodes. A significant number of works [170, 225, 204, 154, 188, 241, 61] have extensively studied application characteristics and proposed architectural techniques to reduce/reuse compute, curb data-flow movement, and reduce memory access. These techniques have definitely improved performance and reduced power consumption. However, they are not yet sufficient to provide a seamless user experience for ever-increasing user applications' computational requirements. For example, the latest smartphones, Samsung's S22 [26] and Apple's iPhone 14 Pro [21], serve maximum video resolution of 8K at 24fps and 4k at 60fps, respectively, whereas upcoming AR/VR applications require high-resolution frames at 90 fps and higher [25, 117]. Another emerging solution relies on the development of application-specific hardware accelerators for specialized applications like machine learning, genome sequencing, neuromorphic computing, etc. However, the accelerator approach is not scalable nor feasible for a multitude of newly emerging application domains. It also requires applications to have well-defined data-flow or compute patterns that can be mapped to customized hardware. Finally, the last approach of shifting to newer/sophisticated silicon technology nodes for better performance and energy is bounded by the slowdown of Moore's law. Moreover, the smaller and newer technology nodes have high production and development costs [54, 15], which is extremely expensive for widespread adoption. The last two solutions are impeded by the high procurement cost of upgrading to new hardware, which also contributes to massive e-waste.

An ideal user edge device should be affordable, secure, easy to use/navigate, and have a cloud server's performance with infinite battery capacity in a form factor that fits in the pocket. Therefore, there is a need for inexpensive hardware-software co-design approaches that can reduce the gap between current user devices and the expected ideal device for emerging applications. **Hence, the goal of this dissertation is to improve the hardware performance, energy efficiency, and cost of edge devices to enhance the user experience** as discussed below. Other factors like design, UI/UX, and OS/software are equally valuable but are beyond the scope of our work.

- **Performance/Quality of Service:** Quality of Service (QoS) or performance for user-applications is directly related to the computational power of the underlying hardware. Edge devices cannot host powerful datacenter style hardware components. They are restricted by their form factor and thermal design power (TDP) limit. Furthermore, the definition of performance/QoS varies with the application. For example, audio/video applications quantify QoS by frames completed per second, whereas the performance of recommendation systems is measured by response time or latency. In this dissertation, we propose solutions based on the application and compute capacity of the underlying edge platform to improve performance and elevate user experience.

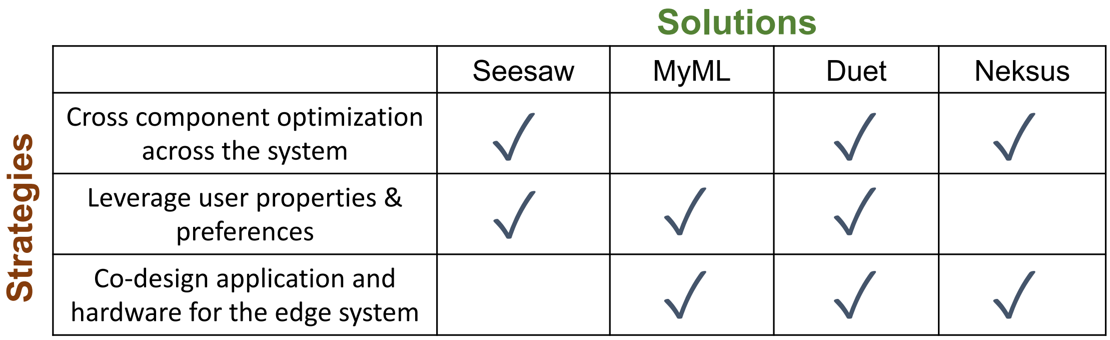
- **Energy efficiency/Battery life:** While handheld devices are getting smaller day by day, at the same time, applications are getting more complex, compute-intensive, and energy-intensive. Unlike cloud servers that are always plugged into a power outlet, edge devices face major battery life concerns owing to their small form factor and portable nature. Furthermore, sometimes the application's nature precludes frequent battery replacement/charging of edge devices, like cameras in wildlife sanctuaries or IoT devices spread across agricultural fields or in manufacturing units. Hence, there is a push to make applications lightweight in order to extend the battery life of IoT/user devices. The proposed solutions in this dissertation consider energy efficiency to be as critical as performance. Our solutions improve energy efficiency while also improving or maintaining performance, thus overall enhancing user experience.

- **Cost/Affordability:** The cost of high-performing hardware is increasing exponentially because of the shift to newer technology nodes and the addition of new application-specific components to chips [136, 15]. Hence, there is a dire need to lower costs to make edge devices accessible to a wider population. There are two ways to reduce the cost of user-devices. The first is to shift to low-cost hardware alternatives without sacrificing power or performance. The second is to transform current user-devices to support emerging resource-intensive user-applications in an energy-efficient manner, which lowers the total cost of ownership of the user device. We provide solutions that reduce the development cost of new chips and also offer techniques to improve user-applications' efficiency on CPU-centric edge platforms that account for most consumer devices deployed in the consumer market.

The first step in developing the work for this dissertation was to find applications that are most relevant to users. Multiple studies [29, 23] conducted in 2022 have found that social media and communications apps (like Facebook, TikTok, Instagram, Messengers), teleconferencing apps (like Zoom, Google Meet), entertainment apps like video streaming and audio streaming platforms (like YouTube, Netflix, Spotify), and online shopping apps (like Amazon, Shopee) are the most popular user apps. Another popular category is gaming apps; however, these have a limited demographic reach. This distribution of mobile apps' popularity was slightly different in 2019, with apps like Uber, Lyft, and Google Maps also being part of the list of frequently-used mobile apps. Maps are also actively used by fitness tracking applications like FitBit and Strava. Another inbuilt application that is very frequently used by consumers on smartphones/smart-cameras is the photo-capture and video-recording feature. In this dissertation, we target these categories since they consume a significant fraction of mobile resources and dominate users' daily activities. A popular legacy application is web browsing. However, it is a very well-studied application, which has been optimized by numerous prior efforts [242, 239, 240, 56, 59, 62, 176, 134] in hardware and software domains that have focused on improving the application's efficiency. Therefore, we do not explore web browsing applications in this dissertation any further.

1.1 Strategies

In this dissertation, we explore various strategies to achieve the above-described goals for emerging edge applications. Primarily, we devise three strategies, which are the foundation of our proposed works, as shown in Figure 1.1. The first strategy is **cross-component optimizations across the edge system**, where multiple components involved in applications work together cohesively to improve the applications' overall efficiency. The second strategy is **leveraging user's inherent**



	Solutions			
Strategies	Seesaw	MyML	Duet	Neksus
Cross component optimization across the system	✓		✓	✓
Leverage user properties & preferences	✓	✓	✓	
Co-design application and hardware for the edge system		✓	✓	✓

Figure 1.1: Strategies guiding proposed solutions.

properties and preferences to customize applications for the underlying hardware of the edge device, thus, improving the application's performance and energy efficiency. The third strategy is **co-design of applications and hardware for the edge systems**, where hardware optimizations are based on the unique software properties of applications.

The first strategy explores the benefits of owning multiple system components spanning an application. The components can be different IP blocks, multiple sensors, or an edge and datacenter comprising the system. Instead of individual components working in a standalone fashion, we look into inter-component optimizations where the components work together and aid each other to complete the given task efficiently. They achieve this by communicating intermediate results directly to each other, sharing essential insights with each other, which helps the other components, and distributing the work in a balanced manner based on the benefits and strengths of each component.

The second strategy leverages the opportunity to tailor apps for the underlying hardware based on individual user properties and preferences. In this dissertation, we first make an observation that the QoS/performance and energy efficiency of user-facing applications can depend on individual users' characteristics/behavior. For example, audio/video applications' QoS depends on the sharpness of the user's sensing capabilities, or GPS coordinates depend on user-chosen routes. Second, we observe that user behavior and properties can also be leveraged by emerging error-tolerant machine learning applications like image recognition, voice recognition, and recommendation systems. These applications are flexible and malleable; thus, individual user traits can be used to simplify the computation and improve the performance and energy efficiency of the execution on edge platforms. We draw upon these insights and explore the potential of leveraging user properties to achieve our goals.

The third strategy co-designs the application and hardware for the edge system. We observe that certain application properties that are visible to software can be leveraged by hardware to optimize the application. For example, there are opportunities for computation reuse and computation sharing for various user applications, which can be exploited by hardware to reduce the operational intensity of an application. Further, we can judiciously balance the compute demands of an application across the edge system, comprised of device and cloud, based on the capabilities of the underlying hardware and the requirements of various parts forming the application.

In the next section, we will present the contributions of this dissertation developed on the three strategies discussed in this section.

1.2 Contributions

In this dissertation, we offer hardware-software co-design solutions to enhance hardware-related parameters that contribute to user experience - performance, energy (battery life), and cost - based on the strategies discussed above. Our target workloads are sub-tasks that are part of popular user-facing applications. As shown in Figure 1.2, these sub-tasks include streaming for entertainment and teleconferencing; recommendation systems for entertainment, social media, and online shopping; video recording for teleconferencing and smart-cameras; photo-capture for smart-cameras; route tracking for maps application, and image/video classification for social media and smart-cameras. Contributions of this dissertation strive to compute these bulky and complex applications on resource-constrained local edge devices in a lightweight manner, to alleviate expensive transmission back and forth between edge devices and the cloud, reduce dependency on a reliable and fast internet connection, and enhance user data privacy.

In our first work [88], we target personal IoT/edge devices like smart-cameras with video

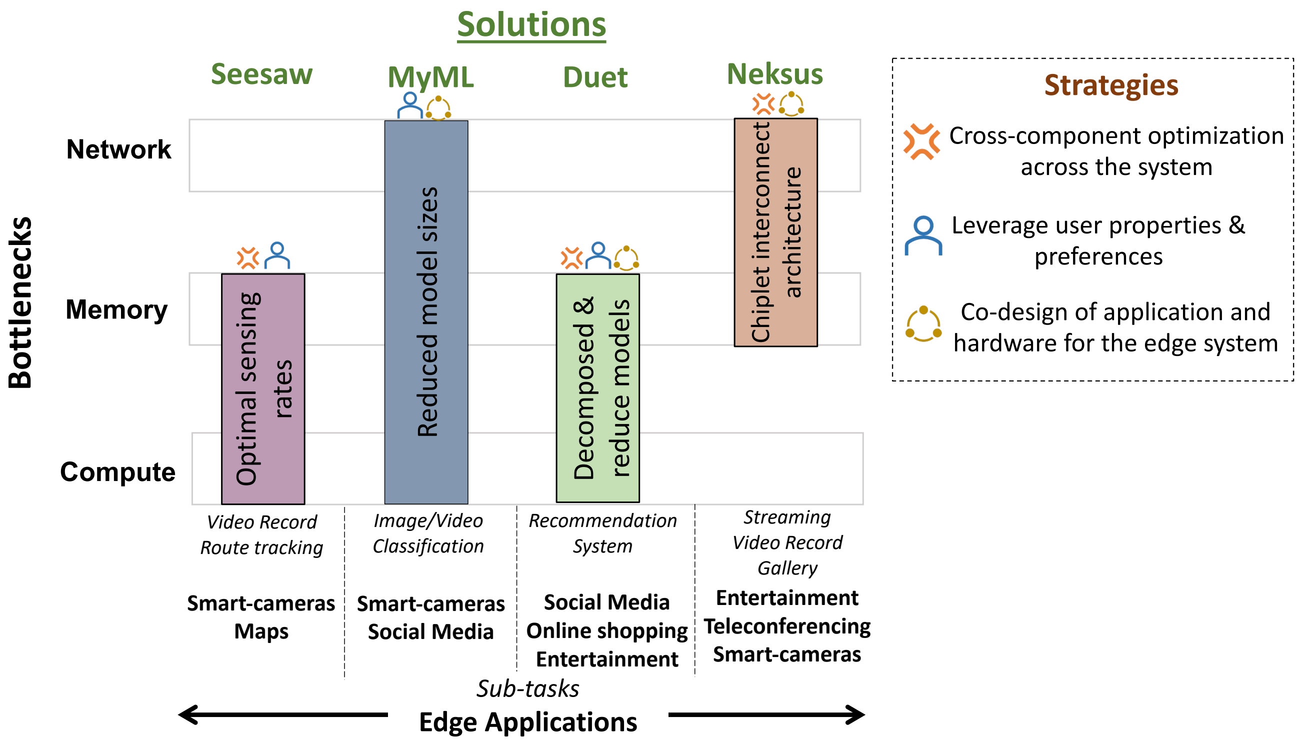


Figure 1.2: Contributions highlighting four works of this dissertation. Each of the works is guided by multiple dissertation strategies and targets a handful of sub-tasks that span across various edge applications. The proposed solutions aim to resolve the three major bottlenecks of any system - Network, Memory, and Compute.

recording and smartwatches/fitness-trackers with the map application. These devices host multiple sensors for various user interactions and are powered by small microcontrollers. We observe that low-power sensors can indicate the impact of high-power sensors' outputs on user experience. When low-power sensors infer that the user is not sensitive to the output of power-intensive sensors, we can dial down the sensing rates of power-intensive sensors and vice-versa. Based on this insight, we developed an end-to-end ML-based solution that automatically identifies correlations between power-intensive and lightweight sensors without human expertise. It deploys a low-overhead decision-tree predictor to determine the optimal sensing rates for power-intensive sensors by using low-power sensors while avoiding significant quality degradation. As depicted by the first (purple) bar in Figure 1.2, the proposed technique impacts the compute and memory overheads by reducing the sensing rate and, thus, the number of data frames processed and stored by the downstream pipeline. This work increases the battery life of edge devices without impacting the human-perceivable experience on multi-sensor edge devices.

Our second work [89, 90] is based on the insight that user behavior can be utilized to increase performance and energy efficiency of handheld user edge devices for the image classification task, related to social media and smart-camera applications, as shown by the second (blue) bar in Figure 1.2. ML tasks, like image recognition, are compute-, memory-, and bandwidth-intensive; thus, they incur high latency and power. The resource-constrained nature of edge platforms further exacerbates the problem. However, ML is a flexible application that can be made lightweight by reducing memory footprint, compute intensity, and bandwidth requirements using the established ML pruning process. In this work, we leverage this malleable nature to prune ML models for user preferences/choices to create small user-specific models. We first learn user preferences and then present a hardware-friendly, lightweight pruning technique to create user-specific models on mobile platforms while simultaneously executing inferences. We also build an end-to-end collaborative system that tracks user behavior changes to create new user-specific models when there is a deviation in user preferences. These small user-specific models have a reduced memory footprint that can easily be computed on edge platforms, thus, improving performance and energy efficiency.

In our third work, we incorporate the user edge device into the end-to-end recommendation system to enhance the performance and energy efficiency of the task. Recommendation task is a critical component of entertainment and social media applications. A generic recommendation model comprises dense multilayer perceptron (MLP) and huge embedding tables with entries for all the possible items in the database, which blows up the table size. It is considered to be a datacenter application because a recommendation query ranks 100s of candidates to recommend only a few, which makes the recommendation task extremely memory and computationally demanding. In this work, we decouple the monolithic recommendation model into user and item models, where we offload the user model in a lightweight manner on the edge device, and the item model is computed on the datacenter. The edge's user model and the datacenter's item model work cohesively to deliver final recommendations. The user model is computed once for a user query on the edge device, and its output is communicated to the datacenter. The edge model output (computation) is then reused across all candidates ranked by the item model at the datacenter. The edge model is processed in a lightweight fashion. It utilizes user information and user history, which is readily available on the edge device, coupled with hardware optimization techniques of memoization, scratchpad, and quantization. These optimizations reduce the computational demand and memory footprint of the recommendation model, as illustrated by the third (green) bar in Figure 1.2. Our proposed edge-cloud collaborative recommendation system reduces the latency and energy consumption of the complete end-to-end application.

Finally, in our last work [91], we design a new interconnect architecture for edge devices to lower the cost and simultaneously improve performance and energy efficiency for video-related tasks of streaming, recording, and photo capturing for entertainment, teleconferencing, and smart-camera applications. We reduce the cost of building new SoCs for user edge devices by migrating from monolithic System-on-chip to heterogeneous System-in-Package (SiP)-based designs, which are comprised of multiple small chiplets bonded together by 2.5D stacking. We show that emerging 2.5D stacking-based SiPs have the potential to reduce the development cost because of the high reuse factor and improved yield of small chiplets over a big monolithic chip. Our proposed solution optimizes data-flow patterns between multiple chiplets by bypassing the memory access and communicating directly. It then maps the data-flow patterns to high-speed and high bandwidth interposer links supported by low-cost 2.5D stacked SiPs. This solution thereby decreases the memory and network overheads resulting in improved performance and energy efficiency, as depicted by the last (orange) bar in Figure 1.2.

1.3 Dissertation Organization

The rest of the dissertation is organized as follows. Chapter 2 discusses the background and related work survey that helped us in developing this dissertation. In Chapter 3, we discuss Seesaw, an end-to-end ML-based approach to improve the energy efficiency of multi-sensor IoT devices. We propose a system where low-power sensors in the device can predict the optimal sensing rates for power-intensive sensors, thus improving battery life without impacting the human-perceivable user experience. Chapter 4 describes MyML, a user-driven machine learning technique where we build user-specific models using a hardware-friendly bottom-up pruning process. The proposed approach leverages compute sharing between pruning and inference, customizes the re-training backward-pass, and chooses the pruning granularity for efficient processing on the edge. MyML drastically reduces model size, thus improving the performance and energy at the edge device. Chapter 5 outlines Duet, which decomposes the monolithic recommendation model into two concurrent smaller models. The user model is computed only once on the edge device, and the item model repeatedly computes all the potential candidates the user may like. Computing only once the user model in a lightweight manner on the edge device offers data privacy and significantly boosts the performance and energy efficiency of the recommendation system. Chapter 6 describes Neksus, an interconnect architecture for cost-efficient System-in-Package, which can replace System-on-Chip, thereby reducing cost. We discover optimized data-flow patterns for audio/video smartphone applications that can be easily mapped to Neksus, resulting in performance and energy improvements. Finally, Chapter 7 briefly concludes this dissertation and outlines future directions.

CHAPTER 2

Background

This chapter reviews the background related to edge platforms. Edge devices are small resource-constrained devices distributed in the environment that are responsible for continuous data sensing, data pre-processing, and data collection. Some very early examples of edge devices are automated toasters and coffee machines, which informed users once breakfast was ready. Since then, they have become more ubiquitous and can be found in wildlife sanctuaries for animal tracking [127], in the depth of the ocean to study marine ecosystems [6], as well as in the hands or pockets of human beings to help them in daily activities. Depending on the use-case and user demand, they come in various shapes, sizes, and functional capabilities, such as RFID tags, thermal sensors, handheld devices, etc.

Historically, they acted as a gateway between sensors and back-end servers, where cloud servers are responsible for the computation of sensed data. However, with the rise in automation, not only is the number of edge devices increasing, but they are simultaneously getting intelligent to support upcoming applications. Thus, it is not sustainable to communicate all the data from edge devices to the back-end server for computation because of limited network bandwidth. This gave rise to edge computing, an emerging paradigm that aimed to move computation close to the data generation for better analytics and reduce network bandwidth requirements. Unlike cloud computing, which offloads computation to back-end servers by transmitting data all the way back to the cloud servers, edge computing computes data locally on an edge device. Computing locally at edge devices eliminates several concerns like the need for reliable internet connection, transmission overheads, longer response time, and privacy. However, the small form factor, limited computing capacity, and portable nature of edge devices still preclude us from adopting edge computing as the default solution for *all* applications, especially emerging compute-intensive application domains like augmented reality, virtual reality, machine learning, audio/video streaming, etc.

The current state of the edge platform is very heterogeneous, which further adds to the problem. Edge devices encapsulate a wide variety of user devices like the internet of things (IoT), smartphones, tablets, autonomous vehicles, etc. They can be as small as a medical implant and as big as a smartphone or tablet. Within the smartphone industry, mobile phones have transformed from a primary device for communication via cellular connection to a source of constant entertainment. This transformation has further increased diversity in edge devices. As per a study [221] by Facebook, there is no standard SoC predominantly used by consumers. They report that the top-50 most common SoCs account for only 65% of the smartphone market, which increases to 225 SoCs accounting for 95% of the market. This diversity and heterogeneity among edge devices and associated applications have made the task of optimizing/improving edge devices for each unique application very difficult. Nonetheless, irrespective of the edge device's nature, the common concerns from a hardware perspective that grip edge devices are the following: 1) development cost that reflects in the final retail cost paid by the consumer; 2) energy efficiency that translates to battery life; and 3) performance that determines the application's functioning on the device.

In the rest of the chapter, we describe prior works that have focused on improving various aspects of edge devices. We first discuss the current state of edge devices, followed by a discussion about popular edge applications and the related prior works that aim to improve the efficiency of these applications on edge devices.

2.1 Edge Devices

As discussed above, there is vast diversity within the ecosystem of edge devices. Anything that can be handheld and powered by a battery can be classified as an edge device. In this section, we broadly breakdown edge devices into two categories: 1) Smartphone or Mobile devices that are powered by relatively powerful system-on-chips (SoCs), such as Snapdragon 855 [27] or A14 Bionic [20], and 2) IoT or Embedded devices that are powered by tiny microcontrollers like ARM Cortex-M4 [5].

2.1.1 Smartphone/Mobile Devices

The number of smartphone subscribers is on the rise and is anticipated to grow to >7.5 billion consumers by 2025 [24] with the potential to grow further in populated regions like China and India. Clearly, mobile computing is here to stay for a long time. Moreover, mobile computing workloads have evolved drastically. They range from conventional applications like voice calls and web browsing to emerging applications like teleconferencing, online streaming, social networking, etc. Because of the increased presence of smartphones and associated applications, there has been significant development in the chipset that powers mobile computing. SoCs that power smartphones now comprise specialized IP blocks along with general-purpose CPU cores, memory, and network components, as shown in Figure 2.1. Two noteworthy trends are dominating the current development effort in the mobile computing domain.

The first is the linear increase in the number of special function IP blocks for new SoCs to support emerging complex applications [199]. This heterogeneity in the SoC landscape requires solutions that improve each component separately and also as an integrated unit. For example, increasing one IP block's performance may not translate to an increase in the application's performance since it may be limited by network or memory bandwidth. Hill et al. [110] is the most recent work that facilitates early-stage SoC design using analytical models to address the issues of portability and high human resources associated with the integration of multiple IP blocks in cycle-accurate simulators like Gem5. It takes into account multiple IPs of an SoC working simultaneously, interacting with memory and network hierarchy. It provides roofline top models to give pre-estimation of SoC performance, which can guide designers in making decisions about which IPs to include, IP size, IP performance, etc. Further, Shao et al.[199] is a simulation-based ap-

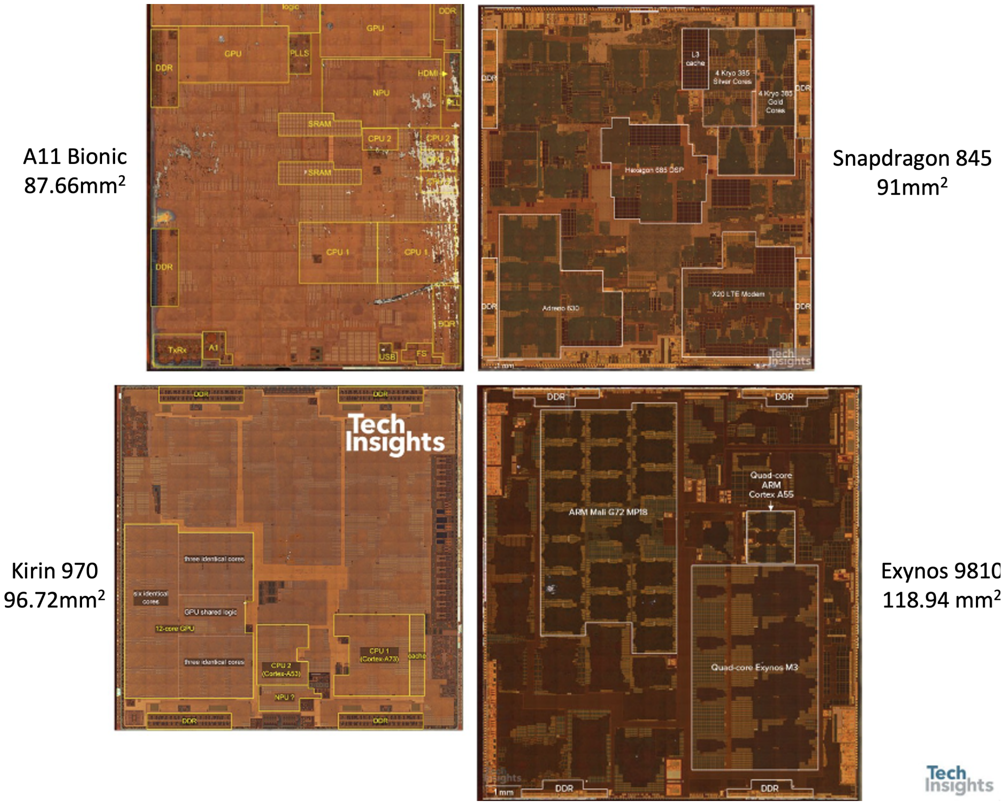


Figure 2.1: Dieshots of 4 majors SoCs [7] from Qualcomm, Apple, Huawei, and Samsung comprising multiple IP blocks.

proach to enable large design space exploration for accelerators/IPs that is otherwise hampered by the slow RTL development process. It gives power, performance, and area estimates given a high-level design (in C/C++) for an accelerator without generating RTL. The work is also integrated with the full system cycle-level simulator Gem5 to give estimates for SoC performance.

While the above works have focused on pre-estimating and improving performance for SoCs with an increasing number of IP blocks, another problem associated with the growing number of IP blocks is the high initial development cost. This is the second problem that has hampered development efforts in the mobile computing domain. As shown by [187], although the number of IP blocks is increasing, the number of total SoC chipsets and the companies developing these chipsets are decreasing. This downfall is accredited to SoCs' increasing development costs because of the shift to expensive technology nodes and high development cost for new IP blocks [136]. It presents significant challenges for hardware startups and, consequently, for the expansion of the silicon industry. For example, a fabless silicon startup may require tens of millions of dollars to bring a product to market, compared to as little as a few hundred thousand dollars for a software startup [15, 54]. The key to the success of the software industry is the wide availability of open-source software infrastructure, which allows cheap reuse of labor. Startup companies can begin with much of their software already in place and then quickly add their own “special sauce” [64].

Hence, there are prior works that have advocated for low-cost heterogeneous System-in-Package (SiPs) based hardware to replace monolithic SoCs. Stow et al. [207, 206] have studied the cost benefits of 2.5D stacking-based chiplet integration over monolithic design owing to yield improvements. [206] also discusses the feasibility of reducing NRE cost using die stacking by leveraging IP reuse across various domains and generations of chips.

Performance study of interposer-based multi-core SiP systems has been done by prior works [120, 131]. Jerger et al. [120] explore the benefits of using silicon interposers for wiring within a multi-core chip. They propose a hybrid Network-on-chip (NoC) lying in the silicon layer, as well as the interposer layer. Memory and core traffic can take advantage of hybrid NoCs, where core-to-core cache coherence traffic is routed through the silicon layer, and main memory traffic is routed through the interposer. [79] discusses the feasibility of network die regarding delay and energy characteristics. Pal et al. [180] show a novel package-less chiplet integration technology—Silicon Interconnection Fabric (Si-IF)—as a replacement for the interposer. There are ongoing works [208, 139] on 2.5D-based SiP packaging, where multiple packaging substrates like ceramic, glass, and silicon interposer properties are being explored. These substrates can be used to support high bandwidth smartphone applications [145]. In the next section, we will discuss IoT/Embedded devices, which is another dominant edge computing platform.

2.1.2 IoT/Embedded Devices

IoT is an emerging paradigm for a range of new capabilities brought about by ubiquitous connectivity and extends computing capabilities to objects, sensors, and everyday items that can exchange data with little to no human involvement [168]. Market growth forecasts that IoT-connected devices will have reached $\sim$11 billion units by 2020 and will reach 30 billion units by 2025 [19].

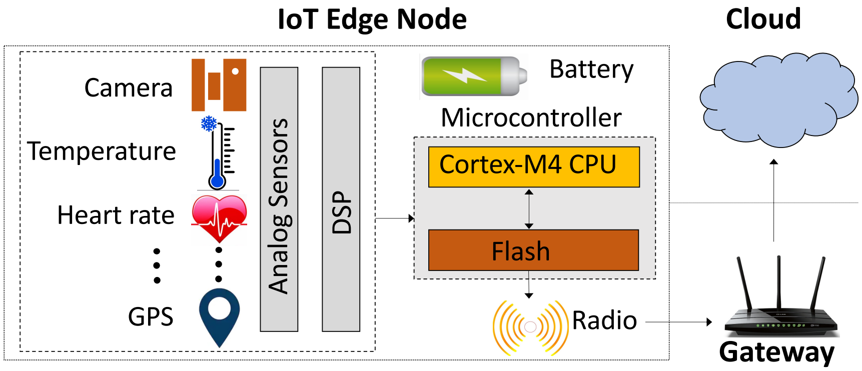


Figure 2.2: IoT platform consisting of sensors, DSPs, microcontrollers, flash storage, battery, and a wireless radio communicating processed data to the cloud via a gateway.

The IoT allows for limitless opportunity: smart cities, smart homes, wearable devices, precision agriculture, eHealth, and factory automation are some of the domains which are adopting IoT capabilities. Some of the popular IoT devices that are found in regular households are personal assistants like Amazon Alexa and Google Home, Nest thermostat, Fitbit tracker, and many more.

At the heart of IoT systems are IoT edge devices, which are capable of continuous data collection and smart networking. Any generic IoT platform consists of edge nodes, gateway nodes, and the cloud. IoT edge nodes are responsible for sensing raw data by using diverse sensors, pre-processing raw data, and communicating it to the cloud via gateway nodes. As shown in Figure 2.2, an IoT edge node typically consists of sensors, a DSP or media processor, a microcontroller, a flash storage, a wireless radio for communication, and a battery. The edge node does lightweight computing on raw data, and the pre-processed data is then transmitted to the cloud to compute the final output. Thus, the majority of time is spent sending data back to the cloud or an edge gateway.

More recent IoT devices like Apple smartwatches [4], Echo Show [3], Google Home [14], and Ring video doorbell [17] have small arm cores to enable higher processing at the device and reduce the data transfer from the edge to cloud. However, the lightweight CPU cores can still only process a small portion of the application because of their limited computing capabilities. Thus, we cannot completely eliminate the transfer overheads between the edge and the cloud. These problems are further exacerbated with the launch of more sophisticated edge IoT devices like Amazon Astro [2], which require more processing power to support many computationally intensive applications.

Many prior works have strived to improve applications' efficiency and have proposed intriguing techniques like developing application-specific microcontrollers [70, 71], intelligently offloading computing to cloud [159, 204], or optimal partitioning of mobile and cloud computation [130, 213]. [71] presents an automated hardware-software co-analysis technique to accurately determine application-specific energy requirement and peak power for low-power processors, and [70] proposes an automated approach that tailors microcontrollers to target applications by removing unwanted gates not used by an application, and thus reduces the power and area. In the next section, we discuss in detail the above and many more prior works that have focused on improving mobile and IoT devices for a wide range of popular user applications.

2.2 Edge Applications

In this section, we will discuss three broad application domains encompassing the most frequently visited user apps on edge devices. The first is computer vision-based applications that process images and videos, such as video recording/playback, photo/video capture, video streaming, and teleconferencing. The second is GPS-based applications, such as map navigation and route tracking. The last is machine-learning-based applications of image classification and recommendation.

2.2.1 Computer Vision

With the boom of social media platforms for photos and video sharing, online streaming services, such as Youtube and Netflix, teleconferencing or videoconferencing apps like Zoom, Google Meet, Messenger, etc., the demand for better QoS and longer battery life has grown drastically. Edge devices are upgrading to high-resolution power-hungry cameras and larger batteries to meet this ever-increasing demand. This trend may not be sustainable in the long run, especially for new emerging applications like Augmented and Virtual reality, which have to run on a small headset and ensure a smooth, immersive experience. Thus, there is a dire need for hardware/software solutions to enhance the overall efficiency of computer vision applications.

Prior works have proposed many interesting solutions to address this problem for mobile and IoT edge devices. On the IoT front, [156] [55] have studied the potential of reducing power consumption in mobile vision applications by changing frequency and moving to low power mode while reducing frame rates. Glimpse [171] is another prior work that focuses on the sub-selection of frames for offloading to the cloud for further heavy-weight vision processing. The selection framework is based on sensors like PIR, audio, or thermal image sensors that detect humans. Prior works [106, 178] propose dynamically changing the display resolution of a smartphone camera based on user-device distance using ultrasonic sensors or activity detection. Yin et al. [227] leverage gesture recognition and human activity detection to save the camera and display energy during photography on mobile phones.

For mobile devices, Yedapalli et al. [225] have studied the loopholes in data-flow patterns in the current SoC design for computer vision applications. They have shown that there is unnecessary indirection involved which consumes memory bandwidth and also affects user experience. Therefore, they propose bypassing memory by forwarding through the memory controller and caches/flow-buffers. This work will definitely improve single application performance, but if multiple applications are trying to use the same IP, the second application may suffer due to a lack of resources. Subsequent follow-up work [170] resolves this inefficiency. [121] proposes a methodology to compute reliability requirements for approximately storing compressed and encrypted videos in video capture devices. The methodology improves energy spent on storage. Another prior work [232] improves the energy efficiency of video streaming by frame-batching along with frequency boosting and leveraging content similarity to reduce memory and bandwidth demands. More recent work [233] proposes two mechanisms to reduce memory and bandwidth consumption: 1) exploit value-similarity in a tile to achieve on-the-fly compression and 2) approximate the video frame early in the imaging pipeline. Another legacy application that is widely adopted by the edge ecosystem is GPS sensing. The next section discusses this application in more depth.

2.2.2 GPS Sensing

GPS used for location tracking is part of many popular edge devices. It is present in sophisticated devices like smartphones, smartwatches, and autonomous vehicles (AVs) to get precise location coordinates utilized by many applications, such as Google Maps and Apple Maps, health apps like Strava to record runs, social media apps to provide recommendations based on geo-locations, and many more. GPS is also found in low-power devices like fitness trackers [44] to track user activity, GoPro cameras [39] for adventure sport, and Wildlife tracking [127]. Nonetheless, it consumes significant energy. In fact, we find that in a fitness tracker, a GPS chip consumes 74% of total system power. This energy-hungry nature is also experienced by users while using maps for navigation. Because of its wide applicability and energy-intensive nature, it has been studied thoroughly by many prior works, as discussed below.

EnTracked [141] presents a technique to turn the GPS module on and off depending on activity detected by an accelerometer. Paek et al. [179] propose turning GPS on and off depending on user history, activity detection, and other location detector modules. Another relevant work by Kjaergaard et al. [140] proposes duty cycling the GPS on mobile devices (smartphones) based on activity detection, distance, and compass headings. This work depends on fixed thresholds and thus is not efficient for fitness tracking devices that are user-dependent and used on a wide range of tracks and trails. Youssef et al. [229] estimate new GPS positions using an accelerometer and compass and synchronize these estimates with the real GPS values at regular intervals. Kim et al. [137] do place detection, activity detection, and path tracking using WiFi, accelerometers, and GPS. Their work switches the GPS on and off depending on the place detection output, i.e., it turns on for place departure and off for place entrance. Prior works [157, 125, 151] also propose to improve GPS accuracy in case of weak GPS signal detection. This is done by dynamically changing the localization method from GPS to cellular or Bluetooth depending on the required application accuracy or integrating GPS with inertial sensors. While GPS sensing is one of the popular legacy applications, in the next section, we will discuss the most emerging application domain now: machine learning.

2.2.3 Machine Learning

Machine learning concepts have been around since the 1990s. LeNet, an early simple convolution neural network (CNN), was developed in 1998 [148]. However, it was not until the last decade that it started gaining popularity and found its way into practical application. In 2012, Google's X Lab developed a machine learning algorithm that was able to autonomously browse YouTube videos to identify the videos that contain cats. This happened at the same time as AlexNet [144], one of the first deep neural networks (DNN), was developed for image recognition. Since then, DNN models have evolved drastically. They have become deeper and wider with much more complex layer structures. Apart from the advancement of DNN models, there were two other drivers that led to the ML revolution. First was the availability of big datasets, like CIFAR [143] and Imagenet [77], required to train accurate large ML models. The second was the increase in the computational capacity offered by the hardware required to train and infer these complex models.

Some of the widespread machine learning applications are image recognition, object detection, keyword spotting, speech recognition, and recommendation. Image recognition and object detection employ DNN models [105, 210, 211], which are comprised mostly of convolution layers (CNNs) followed by fully connected layers. These layers compute thousands of multiply-accumulate (MAC) operations to produce an output, thus making them very memory-, compute-, and bandwidth-intensive. Further, speech recognition, keyword spotting, and machine translation are also based on DNN-based language models like LSTM/RNN [112, 222, 191]. These models involve matrix-vector kernel computation and, thus, are limited by the MAC operations. Finally, emerging DNN-based recommendation models [174] are composed of MLP, and embedding layers, which are compute- and memory-bound, respectively.

Recent years have witnessed an explosion in research on application-specific ML accelerators [69, 67, 76, 236, 109, 78, 126, 215]. Almost all new smartphone SoCs have a dedicated neural processing engine (such as Google Edge TPU [12], Samsung's NPU [31], and the neural engine on Apple's A14 bionic chip [20]) to process compute-intensive and memory-intensive deep neural network models on the user edge device. Despite such efforts, according to the analysis reported by the prior work [98], conducting inference at the edge is not energy- or latency-efficient. There is still a lot of room and need to improve the performance and energy efficiency of ML models on edge devices. Therefore, numerous prior works have proposed exciting solutions to address the limitations of these models on the edge device. Here, we will first discuss works that have focused on accelerating inference at the edge for image and language models. We will then present prior techniques that reduce the model size for the edge device. Finally, we will discuss techniques that have focused on improving the efficiency of state-of-the-art recommendation models.

Accelerating inference at Edge node: Many prior works [115, 204, 241, 61, 188, 214, 130] have focused on efficient machine learning at the edge or mobile devices. [115, 214] have proposed accelerators to improve CNN model processing on edge. [61, 241, 188] utilize input similarity to make machine learning efficient for continuous mobile vision and speech recognition. They utilize motion vectors in a video stream to reuse CNN computation across consecutive frames. They trade off energy and vision quality to improve energy efficiency. [204] developed a system to detect if the incoming images are unseen by the working model at the edge device, sending only the unseen images back to the cloud for progressive training. Previous work [130] has proposed solutions to partition the DNN inference computation at the edge and cloud to reduce data movement. They find that not all layers are compute-intensive, and thus, they compute some layers at the edge device while the remaining layers are computed on the cloud. Similarly, Wang et al. [213] also proposed an efficient technique to partition DNN models into mobile and cloud computation to get overall power savings. RedEye [155] moves some CNN computation into the analog domain to reduce data movement and improve energy efficiency.

Smaller ML models: A recent study [221] concludes that machine learning is carried out on CPUs for most of its users. Therefore, there is a push for efficient machine learning on multi-core CPUs. A common and effective approach to support ML on edge devices is to reduce the model size by pruning ineffectual weights [102, 101]. Many prior works [152, 230, 102, 108, 231, 162, 177] guide which weights and how they should be pruned. [216] has utilized asymmetric pruning for an in-cache acceleration of ML inference by adding a coalescing unit. [230] proposes hardware-aware pruning for CPUs and GPUs. There are works that leverage sparsity in weight/bits or tolerance towards lower precision in DNN and have proposed accelerators for such compressed neural networks [235, 181, 100, 74, 238, 200, 75, 201, 182]. More recently, many works [86, 87, 119, 163, 234] have focused on accelerating the pruning process. Prior works [163, 234] are based on the insight that instead of waiting for a baseline model to be trained in order to prune the baseline model, the pruning process can be moved up and mixed with the training baseline model process. They proactively prune/remove near-zero weights after the first few training epochs under the assumption that near-zero weights will not revive during later training epochs. [86, 87] exploits sparsity in dense DNN training computation and maps them efficiently on general-purpose CPU cores. In addition, [119] reduces memory footprint by proposing a lossless and a lossy encoding scheme for convolution and RELU layer to improve the performance of DNN pruning.

Further, there are works that aim to extract small student models from the baseline teacher model to reduce model size. Knowledge distillation [111] is one such seminal work that changes the objective function to train on soft targets, logits that are inputs of the softmax layer, for a small dataset. FitNet [189] builds thinner and deeper networks based on knowledge distillation to also include hints from the intermediate layers. AMC [107] is an automated technique that utilizes reinforcement learning to provide the model compression policy for mobile devices. Prior works [53, 63] use function preserving network transformations [66] to build new compressed models.

Next, we will describe the state-of-the-art optimization techniques for recommendation systems, an emerging machine-learning application.

Accelerating DNN-based Recommendation Systems: Many recent works have been able to successfully accelerate recommendation inference using specialized hardware like GPUs [95, 146], FPGAs [118, 124], and systolic arrays [96]. Furthermore, many accelerators [69, 67, 100, 50, 186, 198] have been proposed to enhance the performance of dense neural network layers. Furthermore, numerous previous works [133, 184, 165, 52, 128, 146, 217, 223] have proposed solutions to overcome the memory bottleneck experienced by embedding tables with two primary techniques: 1) caching hot embeddings or a combination of frequently appearing embeddings on the CPU to bypass DRAM access, and 2) performing embedding lookups and reduction operation in memory using near-memory processing (NMP). Both of the above techniques have shown significant speedups compared to computing on general-purpose CPUs. Prior work RecPipe [96] proposed a multi-stage model by adding a filtering model before the recommendation model to reduce the candidates ranked by the big recommendation model.

In this chapter, we first reviewed the background of edge devices, where we described two popular edge platforms – Smartphone/mobile devices and IoT devices. Next, we described three important and emerging applications domains of edge devices – computer vision, GPS sensing, and machine learning. In this dissertation, we take insights from these works and build solutions over and above them that improve the performance, energy, and cost of ownership to enhance user experience.

In the next chapter, we describe our first solution, *Seesaw*: an end-to-end machine-learning-based approach for energy-efficient IoT edge devices. IoT devices are composed of multiple sensors with varied power profiles. In Seesaw, we leverage users' sensing capabilities and varied power profiles to reduce the overall energy consumption of applications on edge devices. Seesaw is an automated framework that can be applied to any application involving multiple sensors working together to serve the application.

CHAPTER 3

End-to-End Machine Learning Based Approach for Energy-Efficient Multi-Sensor Edge Platforms

In this chapter, we study techniques to improve the battery life of IoT edge devices without impacting user perception to reach the final goal of enhanced user experience. Instead of hosting a powerful and extremely expensive SoC, IoT devices have multiple economical sensors to do continuous data collection and smart networking, which are powered by microcontrollers [44] or lightweight CPU cores [4].

The small form factor of these IoT edge devices constrains their battery life (size). Further, several IoT domains necessitate these devices to be placed in remote, inaccessible locations, precluding frequent replacements. Thus, to improve the consumer experience for this category of edge platforms, the focus in this chapter is on making them energy-efficient to improve their battery life. A natural method to improve battery life is to reduce the energy requirements of an edge node by reducing the data sensing rates and associated computing overheads. However, naïvely reducing sensing rates can result in significant output accuracy loss, leading to key questions about how to optimize the energy efficiency of the system while maintaining an acceptable output accuracy level.

We make two important observations to achieve our goal of improving battery life without impacting user-perceivable output quality. First, we observe that for these personalized and application-specific edge devices, the quality of output is often dependent on an individual's inbuilt perception capability. For example, minute details/changes in videos, GPS routes, weather monitoring data, etc., may not be captured by a human sensing system. The granularity at which sensors capture data can be tailored to individual users depending on their requirements or inherent discerning abilities.

Second, today's edge devices consist of diverse sensors to cater to user requirements, which can be leveraged to improve the overall energy efficiency of the device. Consider, for instance, structural health monitoring systems: they typically have various sensors with different functionalities and accuracy trade-offs, such as a Vibrating Wire Strain Gauge to measure strain, an inclinometer to measure inclination, and a laser displacement sensor to measure the vertical deflection [183]. Another example is a wearable fitness tracker, such as the FitBit Surge [44], which consists of an accelerometer, magnetometer, GPS, and heart-rate monitor. The diversity in sensors usually results in uneven power consumption characteristics, where a few sensors are power-intensive while others consume a tiny fraction of total power.

Nonetheless, we observe that diverse power profiles of various sensors in the system present an opportunity to improve battery life. At power-intensive sensors, we find that only a small fraction of data computed has a significant impact on its output accuracy, while the rest of the data has minimal impact on accuracy. Furthermore, the lightweight sensors can give indications about the impact on the output accuracy level of the data processing done at the power-intensive sensor.

Hence, in this chapter, we propose *Seesaw*, an end-to-end machine learning-based technique to leverage this correlation between different sensors to improve battery life without degrading the human-perceivable user experience. Battery life can be increased by dynamically changing the sensing rates of the power-intensive sensors depending on the importance of the data being processed. For significant data, the sensing rate should be high to minimize accuracy loss, while sensing rates can be lowered for the rest. A low-overhead machine learning predictor is trained to learn the correlation between the outputs of lightweight sensors and the sensing rate of the power-intensive sensor for a given error tolerance limit. The correlation is established if the cross-validation error of the trained predictor is very low, implying that the machine learning model learns a valid pattern. Once the correlation is confirmed, the *low-overhead* machine learning predictor is deployed on the central microcontroller, where at runtime, based on outputs of lightweight sensors, it predicts the sensing rate of the power-intensive sensor while limiting the output accuracy loss.

We evaluate Seesaw for two use-cases - video recording on a mountable video camera, like GoPro [39], and route tracking on fitness trackers, like FitBit surge [44]. For a mountable video camera, which has a camera and GPS-based speedometer sensor, Seesaw finds a direct correlation between speedometer reading and the frame rate at the camera sensor. High speedometer readings imply high pixel motion between frames, and thus frame rates should be high, and vice-versa. Seesaw also gives a strong relationship between compass headings and GPS sampling rate for fitness trackers. We show the established correlations manifest into battery savings by regulating the high-power sensor based on low-power sensors. Our experiments for 45 testing videos, using Netflix's Video Multi-Method Assessment Fusion (VMAF) [40] metric along with a user study across 17 individuals, show an improvement of 32% in video camera battery life without impacting video quality based on human perception. The fitness tracker evaluated across 100 real GPS routes showed a 66% improvement in battery life, with an average GPS route error of only 2.63%.

3.1 Motivation

Despite a multitude of applications, short battery life is limiting the use of IoT edge devices. A common trend observed in IoT platforms is to embed more sensors to improve the user experience. However, such multi-sensor devices usually have one or more sensors that are power-intensive, while others consume relatively low power. This diversity in sensors presents an opportunity to explore the relationship among these sensors. For example, GoPro added a GPS module [39] in their latest version to give users the additional feature of route tracking. For such a mountable video capture device, the image-sensing pipeline consumes the largest share of the overall power budget. From our experimental setup, we measure that image sensing and processing consume 51% and 31% of the total power, respectively, whereas GPS consumes only 10% of the total power. To reduce the power consumed by the imaging pipeline, we observe that a lightweight speedometer can guide the frame rate of the captured video without affecting the quality of the video recorded. The intuition is that a fast-moving GoPro needs to capture quickly changing scenery around it, while the object motion between adjacent frames is insignificant for a slower pace. The above energy management applies to mountable video capture devices such as Google Glass or GroPro. Further, Arable [42], a precision agriculture platform, hosts many sensors like a Normal Vegetation Difference Index (NVDI) sensor, an acoustic sensor, sometimes an NVDI imaging sensor, and sensors to measure air environmental conditions like temperature, humidity, pressure, and photosynthetic active radiation (PAR). Here, the acoustic sensor for listening to rainfall acts as a portable rain gauge. We observe that it can be sampled depending on environmental sensors like humidity sensors. The intuition is that the probability of rainfall is strongly related to humidity; thus, audio sensors can be sampled depending on the measured humidity or from a combination of other environmental sensors.

As described in the background chapter, many prior works have shown standalone use cases where a sensor can give hints to another sensor of the system. However, in this chapter, *Seesaw*, we present a holistic, automated technique that automatically discovers new relationships between the sensors with diverse characteristics to increase the battery life of the edge device without significant quality loss.

3.2 Seesaw

Seesaw is a generalized, end-to-end machine learning-based technique to reduce the overall power consumption of IoT edge devices comprising multiple sensors. Guidelines to deploy *Seesaw* on any given IoT platform are as follows. First, the platform should fulfill two prerequisites: i) it should contain at least two sensors, and ii) at least one of the sensors should be responsible for a major fraction of the platform's power consumption. Once these prerequisites are met, raw data from the sensors is fed into Seesaw. Seesaw trains low-overhead decision-tree-based predictors for various low-power / power-intensive sensor combinations, as shown in Figure 3.1. The goal

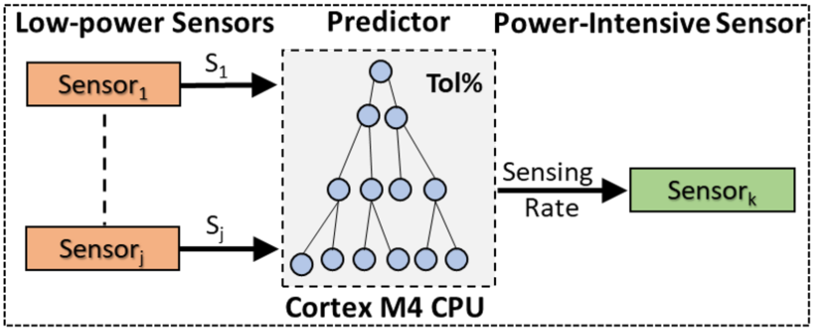


Figure 3.1: Seesaw overview: Low-power sensors predict optimal sensing rate for a power-intensive sensor by means of a low-overhead robust decision tree predictor with low cross-validation error.

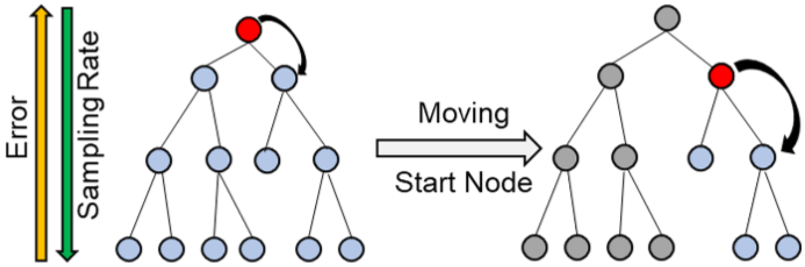


Figure 3.2: Error feedback mechanism dynamically modifies the root node of the decision tree to limit the error.

of the predictor is to regulate the sensing rate of a power-intensive sensor (Sensor$_k$) based on the output values of other low-power sensors (Sensor$_1$, ...Sensor$_j$). Based on the robustness of the predictor, estimated using k-fold cross-validation, Seesaw accepts or discards the correlation learned by the predictor. Only predictors with very low cross-validation error, determined empirically, are accepted, implying that the predictor learned a valid pattern. The predictor is then deployed on the device's microcontroller to modulate the sensing rate of the power-intensive sensor based on information from low-power sensors.

3.2.1 Prediction Model

The predictor must execute on a small microcontroller that should be available on the edge device. Thus, instead of a deep neural network, we use a simple decision-tree predictor to optimize power, performance, and area. The input to the predictor is the output of the low-power sensors, and the predictor's output determines the sampling rate of the high-power sensor. During runtime, the outputs of low-power sensors at time stamp *t* are fed into the low-overhead decision tree predictor, which is extremely fast and runs in a few μs. The predictor processes the input signals provided by low-power sensors to determine the sampling rate of the high-power sensor for time stamp *t+1*. Seesaw works at a granularity of 1 sec; thus, a new sampling rate is predicted every second for the power-intensive sensor. The high-power sensor works at the newly predicted sampling rate for a timing window [*t+1*, *t+2*) until the next sampling rate is predicted at the time stamp *t+2*. At the low-power sensors, only the outputs at the time stamp *t* are fed into the predictor. This can be extended to a value over a timing window of a few secs, for example, a mean of the low-power sensor values for a window of *l* secs [*t-l*, *t*]. A timing window can potentially improve the predictor accuracy but will also add extra computing overhead to the microcontroller. We have not investigated this trade-off in this work and will leave it for future studies.

The decision-tree model is generated for user-defined error tolerance. To train it, we generate data during an offline phase as outlined by Algorithm 1, assuming that the low-power sensors (LP_sensors) guide the sampling/sensing rate of the high-power sensor (HP_sensor). For each training case and each sample within a training case, we sweep the sensing rate of HP_Sensor and measure the sensing error (the lowest sensing rate will generate the highest error) to form an array of error rates along with their corresponding sensing rates. We then create a tuple that maps the sensing rate array and error array, computed in the previous step, to the corresponding low-power sensor output (LP_sensor_out). The global training data is collectively formed by the tuples created for all the training cases and samples within each case. We then use this data to generate a decision tree for a given error tolerance (Tolerance) as described below.

In Algorithm 2, we read the error and sensing rate vectors of each sample (LP_sensor_out). We then choose the index with the largest error rate (sensing rate is minimum) that is still within the tolerance limit. Using this index, we retrieve the corresponding sensing rate for the power-intensive sensor (HP_sensrate_opt). Finally, we map LP_sensor_out to HP_sensrate_opt to form the selected training data, and we train a regression tree model with this data.

Error feedback: Depending on the availability or feasibility of a real-time error measurement technique, an optional error feedback mechanism can also be deployed by dynamically changing the root node of the decision tree. The start node represents the minimum sampling rate (or maximum error) supported by the decision tree. The sampling rate increases from root to leaves, resulting in lower error towards the leaves, as shown in Figure 3.2. For instance, if at any instant the error is greater than the given error tolerance, we move the start node to a lower root node in the decision tree, as shown in Figure 3.2, by traversing the tree and doubling the minimum sampling rate until the error falls within the tolerance threshold. Once the error is stabilized with the tolerance, we explore reducing the sampling rate by a factor of two by traversing back toward the original root of the tree. Note that this adaptive approach based on error feedback is possible because of the inherent structure of decision trees.

Algorithm 1 Generating global training data

```
Input: LP_Sensor, HP_Sensor; Output: Global_Data
for each training case $i$ do
    for each sample $j$ in training case $i$ do
        error ← [], HP_sensrate ← []
        for each HP_sensor sensing rate $k$ in [$K_{min}$, $K_{max}$] do
            local_error ← calc_error (HP_Sensor$_{ij}$, $k$)
            error ← error ∪ local_error
            HP_sensrate ← HP_sensrate ∪ $k$
        end for
        Global_Data ← Global_Data ∪ {LP_Sensor_out$_{ij}$, error, HP_sensrate}
    end for
end for
return Global_Data
```

Algorithm 2 Generating the decision tree

```
Input: Tol (Tolerance), Global_Data; Output: Decision_tree
for each training case $i$ do
    for each sample $j$ in training case $i$ do
        (error[], HP_sensrate[]) ← Global_Data (LP_Sensor_out$_{ij}$)
        (max_error, idx) ← max(error) such that max_error < Tol
        HP_sensrate_opt ← HP_sensrate(idx)
        Training_Data ← Training_Data $\bigcup$ {LP_Sensor_out$_{ij}$, HP_sensrate_opt}
    end for
end for
Decision_tree = regression_tree_model(Training_Data)
return Decision_tree
```

3.2.2 Correlation Finder

We developed a correlation finder to identify any dependency between low-power and power-intensive sensors. Our correlation finder is based on the robustness of the predictor model learned during the offline phase for each pair of low-power and high-power sensors. The correlation finder performs K-fold cross-validation for each predictor model. It splits the training set into K partitions, where one part is selected for testing a model trained on the remaining K-1 parts. This process is repeated for every K$^{\text{th}}$ partition. Root mean square error is calculated for each partition tested, measuring the robustness of the final model trained on the complete training set. The predictor models with cross-validation errors smaller than an empirically determined threshold are accepted as valid correlations.

3.2.3 Seesaw Applicability

As discussed at the beginning of the section, Seesaw is a general technique that can be deployed on any IoT system fulfilling two prerequisites: multiple sensors and diverse power profiles among the sensors. We extensively evaluated Seesaw for two setups (a mountable video camera and a fitness tracker); however, Seesaw can be effective in many other commercial platforms. For example, Arable [42], a precision agriculture IoT platform, hosts many sensors, making it amenable to Seesaw. Plugging in the rain gauge, humidity, and temperature data [45] in the Seesaw method establishes a direct correlation between humidity and acoustic rain gauge sensors based on low cross-validation error (as shown in Table 3.1). Moreover, it also discards any correlation of temperature with the acoustic rain gauge sensor due to high cross-validation error. These correlations align with the intuition that the probability of rainfall is strongly related to humidity but unrelated to temperature. Thus, the acoustic rain gauge sensor, which uses sound classification to calculate the amount of rain, can be sampled at different rates depending on measured humidity.

Similarly, a smart traffic management system that continuously records live videos can use a lightweight passive infrared radiation (PIR) sensor to detect movement in its field of view, which can, in turn, guide the frame rate. As another example, smart parking edge devices [46] use a mag-

	5% Error	10% Error	20% Error
Humidity	0.037	0.032	0.029
Temperature	79.6	71.3	55.6

Table 3.1: Cross-validation errors of models trained for a precision agriculture device.

netometer to detect the presence of a vehicle by computing variations in the earth's magnetic field. Here, naïve optical sensors, detecting the change in intensity of light, can guide the sampling rate of the magnetic sensor, which detects the presence/absence of vehicles in parking spots throughout the day.

3.3 Evaluation Platforms

To evaluate *Seesaw*, we analyzed two specific IoT platforms: a mountable video camera and a fitness tracker.

3.3.1 Mountable Video Camera

The GoPro, a smart video camera, consists of an imaging sensor, image processing DSP, flash memory, Bluetooth module, and a GPS module to collect route information for activities like biking, hiking, etc. Since these commercial devices do not provide fine-grained control to users, which is required to implement Seesaw, we instead replicate the GoPro setup as shown in Figure 3.4(a). On the camera board, we run two image processing filters—a Gaussian and an edge filter—followed by the motion estimation kernel used in video compression to measure active power (P_{act}). We also measure idle power (P_{idle}) and memory write power (P_{wr}). All these parameters are listed in Table 3.2. We calculate the total energy per frame (E_{frame}) and average power at different frame rates using equations 3.1 and 3.2. Here, active time (T_{act}), idle time (T_{idle}), and write time (T_{wr}) will depend on frame rate (number of frames processed per second). Additionally, Bluetooth energy (E_{BLE}) is also added to the final power consumption. Using this setup, we find that the image sensor, image processor, and GPS consume 51%, 36%, and 10% of total power, respectively. The remaining 2% is spent on storage and data transfer. In a nutshell, most of the

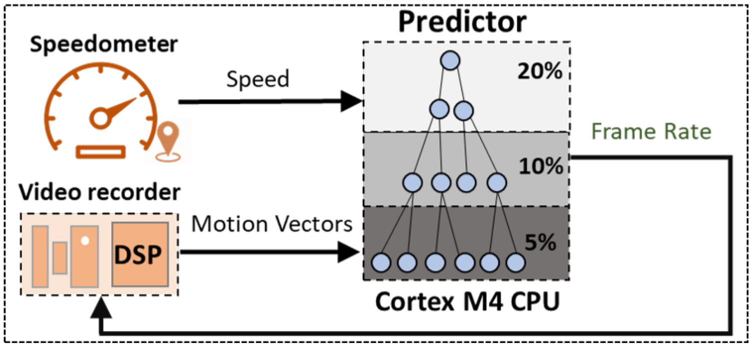


Figure 3.3: Prediction mechanism for Video Camera: Speedometer and motion vectors are used to determine the target frame rate.

power goes to the sensing and computing of images. Thus, energy consumption can be reduced by lowering the frame rate intelligently, with minimal impact on video quality. We validated our power model, based on the replicated GoPro setup, by comparing the battery life from the model with the battery life obtained from running GoPro Hero 5 at 30 fps. We find that the GoPro battery lasts for 2.25 hr, whereas our model estimates a battery life of 2.5 hrs at 30 fps.

$$E_{frame} = P_{act}T_{act} + P_{idle}T_{idle} + P_{wr}T_{wr} + E_{BLE}Frame\_size \tag{3.1}$$

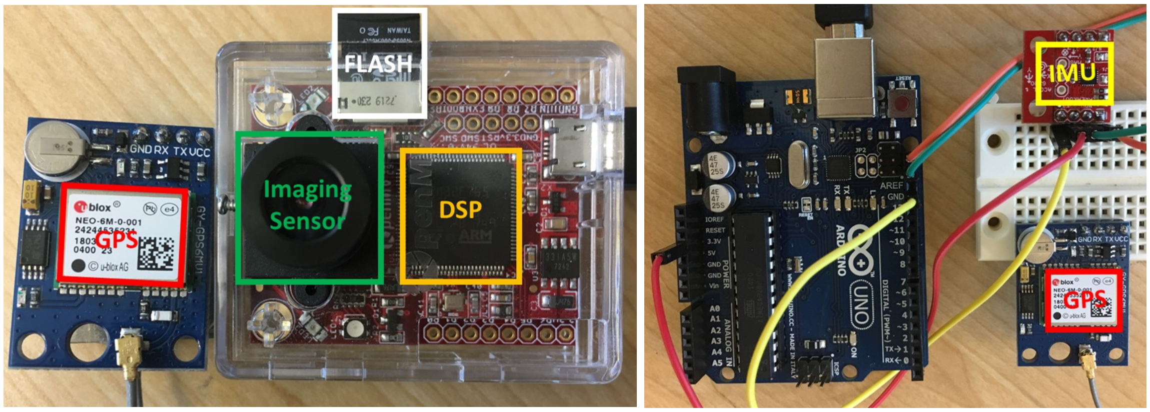
$$P_{fps} = E_{frame}fps \tag{3.2}$$

$$T_{battery} = Battery\_Capacity/P_{avg} \tag{3.3}$$

Prediction Model: The predictor model considers GPS-based speedometer readings to predict the lowest frame rate that produces an error within user-defined error tolerance. The correlation is established only if the predictor model has a low cross-validation error. In addition to speedometer readings, we also provide motion vectors to the predictor as another implicit sensor data. Motion vectors computed for video compression, using a block-matching algorithm, give an estimate of the displacement of objects in subsequent frames. In *Seesaw*, we leverage this information to predict a desirable frame rate. Since the block-matching algorithm bounds its search space to reduce the amount of on-the-fly computation, it does not produce accurate outputs when the camera is moving fast. Hence, the speedometer and motion vectors are complementary to each other and are both used by the predictor model, as illustrated in Figure 3.3.

We used 50 video sequences for training, and another 45 for testing, using GoPro videos available online. These videos belong to one of three categories: low, medium, and high pixel motion. Speedometer data is obtained using the optical flow [82] algorithm. Libraries from the OpenCV framework are used to gather speedometer and motion vectors. The FFMPEG utility is used to process and change video frame rates. We trained the decision tree for three error tolerance values: 5%, 10%, and 20%. To extract the training data, we sweep the frame rate for every second of each video from high to low and create a database, mapping frame rates to the error in the video quality metric defined by Netflix's Video Multi-Method Assessment Fusion (VMAF) [40]. VMAF has three components: visual quality fidelity (VIF), detail loss measure (DLM), and pixel motion. VIF and DLM measure loss in image quality, whereas pixel motion captures errors due to changes in frame rate. The final VMAF output is a fused score of the three individual scores. Since we modify only frame rates and do not alter image quality, the VMAF metric measures video quality in our setup. The lowest frame rate, which produces the highest power savings (but is still within the error tolerance), is selected to train the decision tree. Only models with cross-validation errors within the empirically determined threshold limit are accepted. Bounded by the GPS (speedometer) update rate of 1Hz, the decision tree is invoked every second. The optional error feedback mechanism could not be applied for this application because of the lack of any real-time technique to estimate error. Note that since the VMAF calculation is compute-intensive, the VMAF score is only computed while training the decision tree and not during video capture.

User Study: We conducted a user study of video quality evaluation to relate the error thresholds with human perception of video quality. The user study was reviewed by the University of Michigan Health Sciences and Behavioral Sciences Institutional Review Board (HUM00149764) and was deemed exempt. The study was advertised among graduate students in the department of Computer Science and Engineering. Graduate student volunteers that signed up for the study were asked to read the study description and consent form. They were then given an opportunity to agree or decline. On their acceptance, we presented different versions of two unique videos



(a) Video camera

(b) Fitness tracker

Figure 3.4: (a)Video camera setup comprising OpenMV camera platform and ublox GPS. (b) Fitness tracker setup comprising a Neo-6M ublox GPS and an IMU sensor fusion chip: the MPU9250 configured by Arduino-Uno board. Power is measured using a USB power-meter.

through a laptop to each participant. Each unique video had four versions - one original, and the remaining three were the video outputs from Seesaw's prediction model for three error rates - 5%, 10%, and 20%. After completely watching the first set of videos on a laptop, the participant was asked to complete the first portion of the study via a google form. We then presented the second set of videos, and the participant completed the evaluation on the same google form. The google survey questionnaire asked the participant to provide a subjective comparative evaluation of pairs of videos by providing a numerical score of relative quality. For each portion of the study, the user was asked to compare the three versions of the video with a "baseline" and rate them on a scale of [-10, +10], where positive scores indicate better quality than baseline and vice-versa. To avoid bias, the "baseline" was chosen at random from the four versions and was not necessarily the original video. Furthermore, the users were not informed which video was the original. To further get rid of any bias, we sometimes replicated videos, i.e., different versions might be identical, and users might also receive the same unique videos. This study was carried out for 30 sets of videos picked out randomly from 45 videos consisting of fast videos and slow videos. User inputs were anonymized. The survey also asked the participant whether they are avid video games because we believe those participants are likely to have developed a strong skill in perceiving small differences in video quality. A total of 17 graduate student volunteers, including all genders, completed the

Table 3.2: Video camera

P_{act}	492 mW
P_{idle}	40 mW
P_{wr}	216 mW
BLE energy	0.5 μJ/bit
Battery capacity	5.36 Wh
Frequency	216 Mhz
Frame size	1080p
Frame rate	30 fps
Video compr. ratio	1:15

Table 3.3: Fitness tracker

P_{active}	196 mW
P_{idle}	90 mW
P_{write} [92]	6 pJ/bit
BLE energy	0.5 μJ/bit
Energy Capacity	0.37 Wh
GPS Data Size	8 bytes
Sensor Fusion MPU9250 power	10 mW

user study. Out of 17, 4 identified themselves as avid video gamers with sharp vision. We ensured that volunteers had no information about the optimization techniques we proposed to get unbiased results. Data collected was stored in cloud storage via google drive and processed and aggregated directly on the corresponding data file at the end of the study. The findings of this user study are reported in Section 3.4.1.

3.3.2 Fitness Tracker

Fitbit Surge, a popular fitness tracker, contains an inertial measurement unit (IMU) with a digital motion processor, GPS, Bluetooth, NOR flash, and a central microcontroller. Due to the lack of availability of fine-grained control on commercially available devices, we replicated a fitness tracker setup for our evaluation, shown in Figure 3.4(b). The GPS chip performs correlation-based FFT search and transmits the output in the form of the National Marine Electronics Association (NMEA) sentences to the Arduino board, which performs parsing and distance calculation. The three operations – FFT, parsing, and distance calculation – contribute to active power. All the parameters are listed in Table 3.3. The analytical power model is similar to that of the mountable video camera, which used equations 3.1, 3.2, and 3.3, where the frame rate (fps) is replaced by the update rate, and T_{act} and T_{idle} depend on the update rate. Using this setup, we find that GPS and IMU consume 74% and 21%, respectively, of the total system power. Other components together accrue 5% of the total power. Thus, the GPS sampling rate or update rate should be reduced intelligently to improve battery life. We compare the battery life from the power model of our

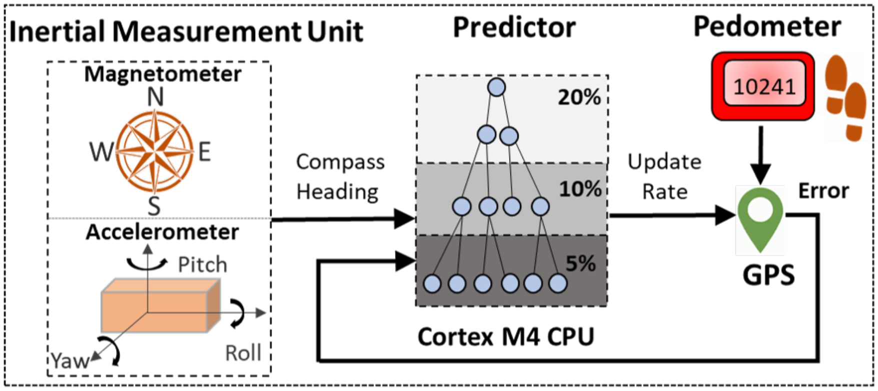


Figure 3.5: Prediction model for a fitness tracker. Error tracked using distance measurement from pedometer and GPS is used as feedback to change the decision tree dynamically.

setup with the optimistic battery life reported by Fitbit Surge [43], where only GPS is active, and IMU, MCU, *etc.* are excluded. Fitbit reports at most 10 hrs of battery life, while we measure Battery life of 7.8 hrs from our setup with GPS, IMU, and MCU all active, at an update rate of 1 Hz – a fair approximation of the Fitbit device's reported value.

Prediction Model: We trained a predictor model that takes compass headings from the lightweight IMU unit as input to predict the optimal GPS update rate within the given error tolerance, as shown in Figure 3.5. If the predictor model has a low cross-validation error, the correlation is established. The update rate is swept from 1Hz to 0.1Hz for each GPS route to generate the training data that maps the update rate to the route error, defined by equation 3.4. The lowest update rate within the given error threshold is selected to train the decision tree. We used a total of 200 real GPS routes (100 train and 100 test), which are uploaded by users on Kaggle [41], to train our model. Decision trees are trained for three error tolerance limits – 5%, 10%, and 20% – and the one in use is invoked every second. Only prediction models with cross-validation errors within the empirically determined threshold limit are accepted.

$$Route\_error = \frac{\Sigma(d_{measured} - d_{original})_t}{Total\_distance} \tag{3.4}$$

Error feedback: Real-time error tracking is feasible, and it is based on the difference between the distance calculated by GPS and the distance computed by the pedometer, using step count and stride length. This error is fed into the decision tree to dynamically change the minimum update

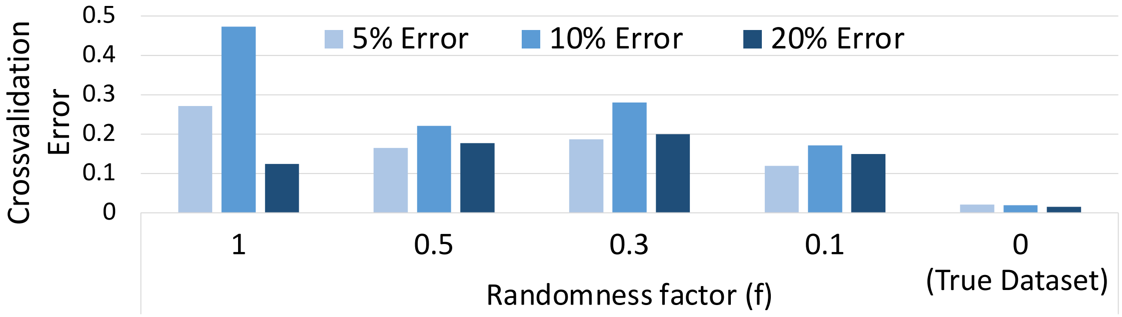


Figure 3.6: Cross-validation errors of decision trees trained for the video camera over a range of error tolerance limits.

rate (pointed by the start node) that it can support, which reduces the final output error as discussed in Section 3.2.1. This closed-loop mechanism adapts the predictor model irrespective of the user and terrain to limit the final output error.

3.4 Experimental Evaluation

3.4.1 Mountable Video Camera

Established Correlation: The K-fold cross-validation (k=10) method revealed that, for *true dataset*, the cross-validation error is of the order of 0.01, as shown in Figure 3.6. For the rest of the randomized datasets, wherein f represents a fraction of random data (f>0), the error is at least 10× larger (> 0.1), and the error increases as the data diverges from the true dataset. For the true dataset, Seesaw established a direct correlation between the frame rate of the camera and the speedometer and motion vectors. The speedometer can be used to control the frame rate of the video pipeline. During high-speed adventures like biking, rollerblading, or free-fall, the frame rate should be high to capture all activities, but for regular activities, such as walking or diving, a lower rate is acceptable. Another scenario that requires high frame rates occurs when objects are moving rapidly around a still camera. Such a scenario is captured by motion vectors calculated for the video compression task. Thus, this correlation, where the speedometer and motion vector guide the frame rates at the camera, is used for the rest of the evaluation.

Frame Rate: Figure 3.7 shows the change in frame rates for 45 videos evaluated. The videos are

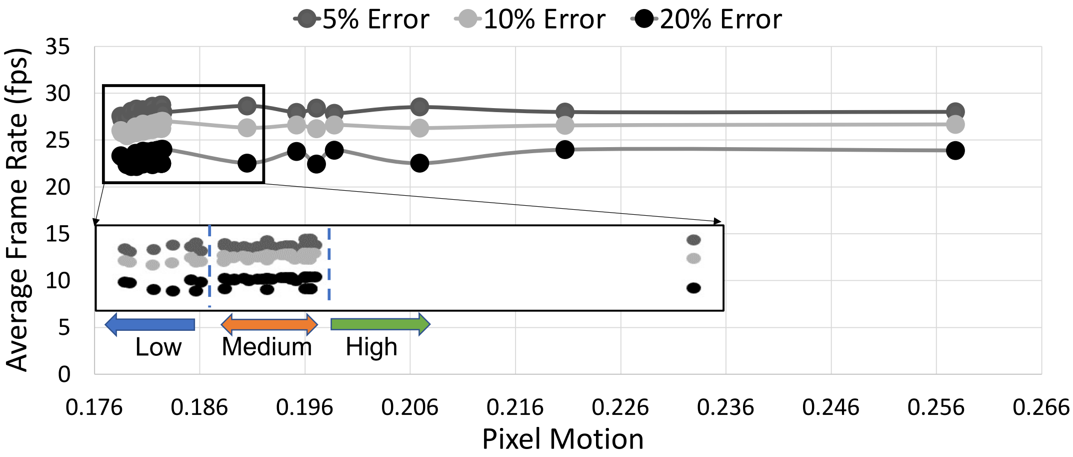


Figure 3.7: Average frame rate across 45 videos for different error tolerance limits. Videos are sorted based on their average pixel motion (12 low, 26 medium, and 7 high).

sorted by average pixel motion. As pixel motion increases, the delta between frames also increases, indicating a rise in activity factor. Frame rates decrease as the error limit increases to accommodate more errors. The average frame rate is 28fps, 26.4fps, and 23.4fps for decision trees trained at 5%, 10%, and 20% error tolerance, respectively. We also observe that the frame rate is lower,*i.e.*, the predictor can drop more frames, when pixel motion is small. Furthermore, there are fewer frame drops (the frame rate is higher) for more dynamic videos to retain significant data intact.

Video Quality: Figure 3.8 shows the average VMAF error for each video for the three error tolerance limits. Videos with high pixel motion have low errors, so no significant information is lost. For slower and smoother videos with low pixel motion value, a larger error can be tolerated since there is additional redundant information, the loss of which does not impact the quality of the user's experience. The average error is 4.9%, 7.7%, and 16.5% for 5%, 10%, and 20% error tolerance, respectively. Thus, the average error is always within the error tolerance and increases as the error tolerance increases.

User Study: We also relate these quantitative errors with our results from the user study to capture human perception. The user study for carried out rigorously for 17 users. The goal of this user study is to understand at what error tolerance limit humans can perceive a major difference with the naked eye. We aim to find which of the three error tolerances—5%, 10%, and 20%—are acceptable. Average scores from the user study are listed in Table 3.4. The users had to assign scores

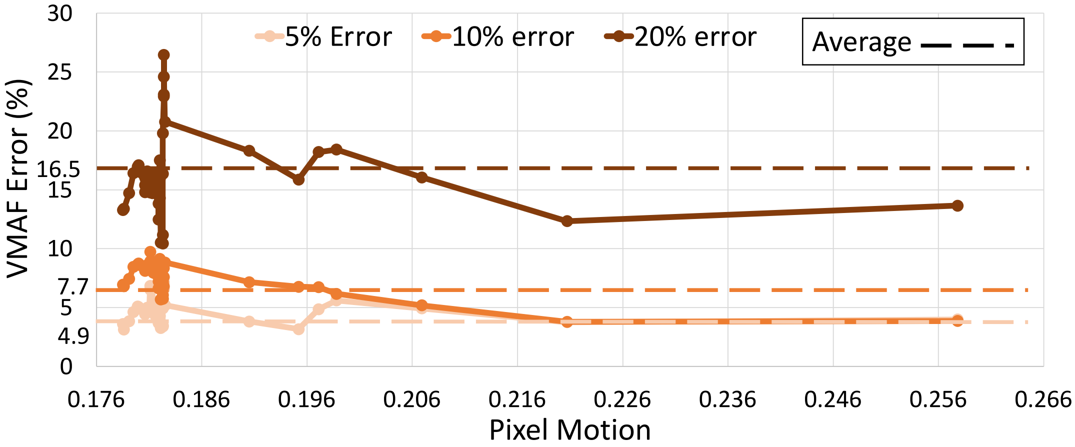


Figure 3.8: VMAF video quality error measured across 45 videos.

between [-10, 10]. Users were not informed which version was the original video, what decision tree error thresholds were used for different video versions, and sometimes videos were replicated to eliminate bias. Thus, users did not always rate the original video as the highest quality. It can be seen that all the average values are close to 0 and acceptable. For slow videos, the difference is almost negligible since there is not much information loss by getting rid of redundant frames. In fact, the value is positive for 10% & 20% error since it tends to create a smoothening effect. For the medium pixel motion range, the average score decreases with a higher error tolerance limit. Hence, users can see some small changes, but the overall quality is acceptable. For high pixel motion with fast-moving objects, it is difficult for users to track what was intact and what was lost; hence we see irregularities, but the absolute scores are very close to zero. Post-processing methods such as video motion interpolation, which can be combined with video decompression at the back-end, can also be applied to enhance the user experience; however, we did not explore this option for our work. Hence from this study, we conclude that intelligently changing frame rates has minimal impact on user perception. Thus, we can safely choose 20% error tolerance or more as an acceptable error limit. It can be higher than 20%, but we did not explore the option in our user study and left it as future work.

Energy Efficiency: The average increase in battery life is shown in Figure 3.9. Our energy model accounts for the entire image sensing pipeline (analog and digital), video data transfer, speedometer sensing/computation, and decision tree predictor. We show two baselines: one without sleep mode

Pixel Motion	Original	5% Error	10% Error	20% Error
Low pixel Motion	0.5	-0.25	0.25	0.062
Medium pixel Motion	1.20	-0.23	-1	-1.2
High pixel Motion	0.5	-0.8	0.8	-0.2

Table 3.4: User study scores averaged across 45 videos for 17 unique users. All the average values are close to 0 and acceptable.

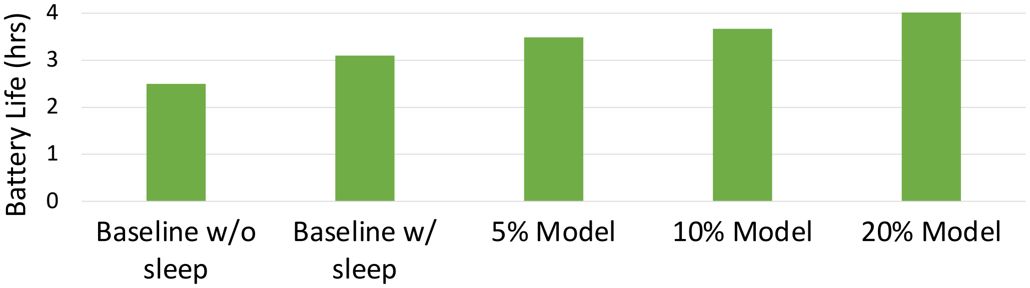


Figure 3.9: Average battery life of camera across error tolerance models.

on (Baseline w/o sleep), during which the processor is always on; and one with sleep mode on (Baseline w/ sleep), during which the processor switches to sleep mode in between frames when it is not being used for any computation. With sleep mode enabled, battery life improves by 18% at 30 fps. On using Seesaw, the battery life improves by 12.5%, 18.3%, and 31.5% for 5%, 10%, and 20% error tolerance, respectively, over a baseline with sleep mode. Savings improve as the error limit increases because the predictor allows more errors and more frame drops. We also observe that energy savings are higher for low-pixel motion or slow videos since the average frame rate is low, and vice-versa.

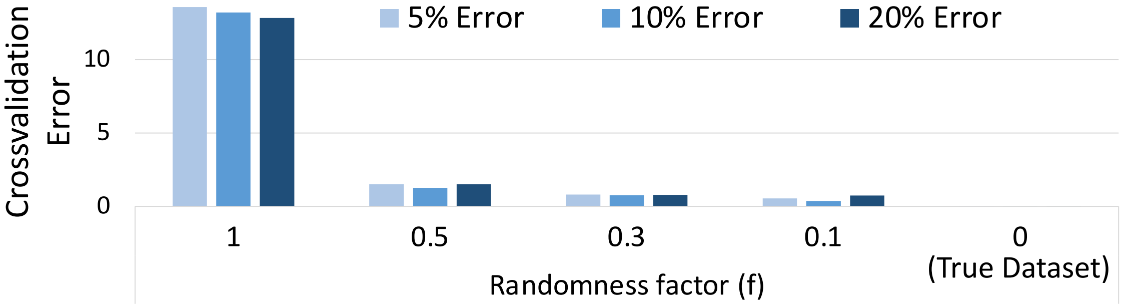


Figure 3.10: Cross-validation errors of decision trees trained for the fitness tracker.

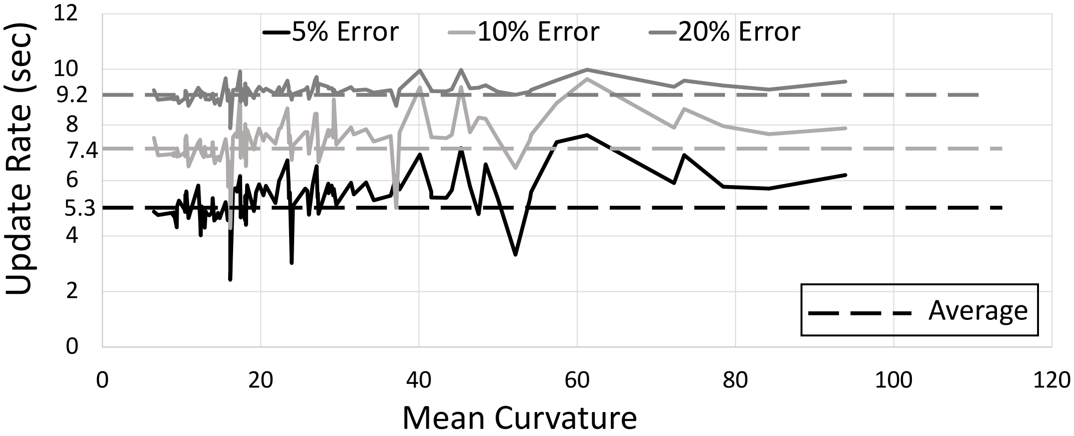


Figure 3.11: GPS update rates across 100 routes sorted based on their average curvature. The update rate is measured as the time between two updates. In the baseline system, the GPS updates every second.

3.4.2 Fitness Trackers

Established Correlation: On applying K-fold (K=10) cross-validation method, we find that the cross-validation error is ∼10E-02 for the true dataset, as shown in Figure 3.10. For the rest of the randomized datasets, where f represents the fraction of random data, the error is at least 10× and, at most, 100× higher than the true dataset. For the true dataset, Seesaw established a direct correlation between the update rate of GPS and compass heading from the IMU sensor fusion chip. Compass headings indicate a change in direction on the GPS route, which can be used to control the update rate of GPS since the route or distance information does not change until a turn is present on the GPS route. This correlation, in which the IMU compass heading guides the GPS update rate, is used for the rest of the evaluation.

GPS Update Rate: Figure 3.11 shows the update rate across 100 testing routes. The routes are sorted based on their average curvature, a measure of the irregularity of a route. Routes with lots of twists and turns, *e.g.*, hiking trails, will have a higher curvature as compared to routes with straight paths, *e.g.*, running across a paved city road. The average update rate is 5.3s, 7.4s, and 9.2s for our 5%, 10%, and 20% error tolerance model, increasing as the error tolerance increases.

Error: Figure 3.12 shows the maximum error across 100 routes with and without the error feedback mechanism. The maximum error with error feedback is 5.03%, 9.35%, and 17.16% for our 5%, 10%, and 20% error tolerance model, which increases to 17.44%, 23.12%, and 24.93% with-

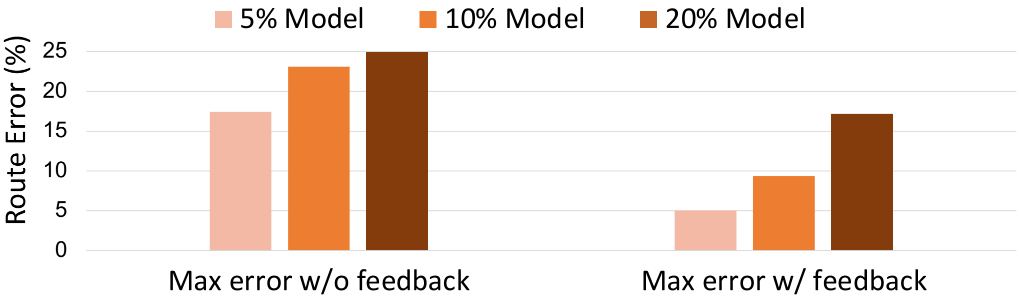


Figure 3.12: Fitness tracker maximum error across the routes with and without feedback mechanism.

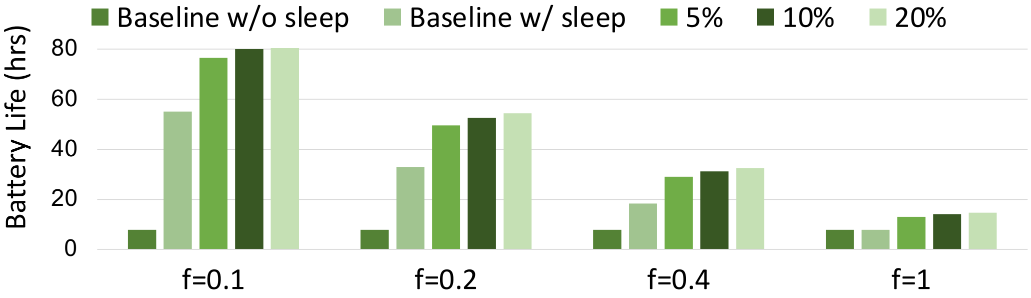


Figure 3.13: Fitness tracker battery savings across activity factors (f).

out the error feedback mechanism. For the closed-loop error feedback mechanism used by our proposed approach, the decision tree changes dynamically to adapt to any route; thus, the maximum error is within the tolerance limit. For both, with and without error feedback, the average error is 2.63%, 4.02%, and 5.34% for 5%, 10%, and 20% error tolerance, respectively. As expected, average error increases as error tolerance increases.

Energy efficiency: The average increase in battery life is shown in Figure 3.13. We report results for different activity factors, which capture user active time, *i.e.*, the fraction of time in which the GPS is actively used. We show battery life savings for four activity factor (f) values: 0.1, 0.2, 0.4, and 1. We also show two baselines: a baseline without sleep mode on (Baseline w/o sleep) and a baseline with sleep mode on (Baseline w/ sleep). Here, Baseline w/ sleep also resembles work that turns GPS on/off depending on some activity detection, which is represented by an activity factor (f).

On enabling sleep mode, the battery life increases by 7×, 4.2×, and 2.3× for activity factors (f) 0.1, 0.2, and 0.4 as compared to always-on mode. As the activity factor increases, this difference reduces, and the two baselines become equal for f=1 when GPS is always actively used by the consumer. With Seesaw, we observe that even with the smallest activity factor of 0.1, battery life improves by 39%, 45%, and 49% for 5%, 10%, and 20% error tolerance, respectively, w.r.t. the baseline w/ sleep mode enabled. For a maximum activity factor of 1, battery savings shoot up to 66%, 79%, and 88% for 5%, 10%, and 20% error tolerance, respectively. Our savings increase as the activity factor increases since more GPS activity means more opportunities to optimize. Savings also increase as we tolerate more error since the average update interval increases, allowing the processor to sleep longer.

3.4.3 Prediction Model Overhead

To estimate the energy overhead of the decision tree, we run the model on an ARM M4 controller [36]. The average decision tree power for the two use cases—mountable video capture and fitness trackers—is shown in Table 3.5. It is insignificant: 0.210μW and 0.610μW for the fitness tracker and video capture, respectively, which amounts to 0.1% of the power consumption of the other components in these devices. Note the prediction model is invoked every second and runs for 1.043μs for the fitness tracker and 2.83μs for video capture. The decision trees have, in total, 17 nodes for the fitness tracker and 19 nodes for the video capture. The two decision trees take 60 kB of 1024 kB and 20 kB of 256 kB available flash and RAM, respectively. This is only 7% of available memory, and thus *Seesaw* can be scaled to accommodate more decision trees generated for different applications. Decision trees are used because they are lightweight and perform reasonably well. We also did an evaluation with a Support Vector Machine (SVM) as the predictor to observe the trade-off between accuracy and predictor overheads. Using SVM with Gaussian kernel, we found that the power overheads increased to 849 μW and 73 μW for the fitness tracker and video capture, respectively, which is >100x larger than overheads of the decision tree. Moreover, we did not find any improvement in accuracy with SVM. The SVM model resulted in the same application error as found with the decision tree but at a higher power overhead of SVM.

	Decision Tree	SVM
Fitness Tracker	0.211 μW	849 μW
Video Camera	0.610 μW	73 μW

Table 3.5: Power overhead of decision tree and SVM.

3.5 Conclusion

In this work, we leveraged the relationship among sensors with diverse power profiles to increase the battery life of edge devices. We presented *Seesaw*, an end-to-end machine learning-based solution that used a low-overhead decision tree model to automatically identify correlations between high-power and low-power sensors. Specifically, Seesaw explored whether a low-power sensor can determine the impact on output accuracy by the data being processed at the high-power sensor. Upon establishing a correlation, it used a decision-tree predictor that, based on low-power sensor outputs, predicted the best sampling rate for the high-power sensor within a given error tolerance. We assessed the benefits of Seesaw for two case studies: (1) video recording on a mountable video camera, like the GoPro, and (2) route tracking on fitness trackers, like the FitBit Surge. We showed that the established correlations improve battery life for the mountable camera and the fitness tracker by 32% and 66%, respectively, without any significant accuracy loss. In this chapter, we leveraged humans' implicit discerning capabilities to tailor the application for each user to improve the battery life of personal IoT/edge devices. In the next chapter, we will leverage users' explicit preferences/choices to optimize the emerging machine learning-based computer vision application for the underlying hardware to improve the consumer experience.

CHAPTER 4

User-Driven Lightweight Machine Learning for Edge Devices

In the last chapter, we presented a technique to tailor the input sensing rate of applications based on users' implicit sensing capabilities. In this chapter, we leverage user preferences to tailor the application kernel to individual users on their personal edge devices. We broaden our application set to include emerging machine learning-based image/video recognition tasks, which is the most popular use-case of machine learning in computer vision, to attain our goal of enhancing the user experience on edge devices.

Machine learning on resource-constrained edge devices with multi-core ARM CPUs is computationally expensive and often requires offloading computation to the cloud. However, the type of data processed at edge devices is user-specific and limited to a few inference classes. In this work, we find an opportunity to build smaller, user-specific machine learning models based on user preferences rather than utilizing a generic, compute-intensive machine learning model that caters to a diverse range of users. Based on this insight, in this chapter, we present MyML, a hardware-software approach to make machine learning on edge devices more feasible. We leverage transfer learning [228] to create small, user-specific models based on user preferences instead of defaulting to the complex original model. Transfer learning is an approach to learning models for a new domain by re-training the currently available models with new domain inputs. We draw upon this insight to build small user-specific models by simultaneously pruning and re-training the current original model *locally* at the user device in an efficient way, feasible for resource-constrained edge devices.

For MyML, We first developed a hardware-cognizant software solution to create user-specific models without sending user data to the cloud. We propose a hardware-friendly, bottom-up pruning scheme, which utilizes the unique opportunity of simultaneous inference and pruning to share computation between the two. In bottom-up pruning, we prune one layer (or group of layers) at a time and start pruning from the last layers of the model, moving up to the top layers. Bottom-up pruning utilizes a structured pruning approach to achieve high training efficiency on edge CPU and edge accelerator platforms. This work explores two kinds of structured channel pruning, symmetric and asymmetric pruning, that have different trade-offs between pruning rates and pruning granularity. Symmetric pruning works at coarse pruning granularity, leading to a lower pruning rate, but it does not need a fine-control mechanism and the related overhead. On the other hand, asymmetric pruning prunes at a finer granularity, thus, yielding a high pruning rate. However, asymmetric pruning requires a sophisticated bookkeeping control mechanism for fine-grained computing, which has a small overhead. Based on the properties of the underlying hardware, we show that Edge accelerator platforms, like Edge TPU [12] with the 2D systolic array, can support symmetric pruning. In contrast, asymmetric pruning can be enabled at edge CPU-only platforms, supporting a fine-grained bookkeeping control mechanism.

We show that, for the widely accepted Resnet-50 model, our user-specific model for five user classes is 4.3× smaller and has comparable accuracy ($\leq$1% accuracy drop) to the original ML model while speeding up inference by 2.9×. For the more complex Inception-V3 model, our user-specific model for five user classes is 4.7× smaller and has comparable accuracy ($\leq$1% accuracy drop) to the original ML model while speeding up inference by 2.3×. Our first sensitivity study is for per-layer pruning and learning rates. We show that the bottom-most group of layers is the major contributor to the model size and has the highest pruning rates of 78%. The pruning rates gradually drop as we move to the top layers while stabilizing the accuracy. On the learning rate front, the bottom layers have higher learning rates to facilitate initial fast learning. The learning rates then drop slowly for the top layers for a stable and accurate model. The second sensitivity study on training batch size, which determines the size of the dataset required to train a model, gives an optimal batch size of 8 with the best trade-off between dataset size, accuracy, and model size. The largest batch size of 64 with a much larger dataset did not offer significant benefits in model size and accuracy. Our last sensitivity study, where we increase the number of user classes, shows that our approach is scalable to a wider set of user classes representing an expansion of user preferences, resulting in a model reduction of 3.2x for 40 classes. Furthermore, our bottom-up pruning technique can converge to a user-specific model by processing 200 images per class at a pruning/training throughput of 2.94 images/sec and 2.56 images/sec for ResNet-50 and Inception-V3, respectively, on the octa-core Snapdragon mobile SoC.

Further, we develop a collaborative system that computes ML inferences at the edge using the user-specific model and tracks changes in user preferences based on prediction probability and entropy over probability distribution. Based on the estimated divergence in user preferences, it determines when to discard the current user-specific model and bring back the original model to restart a new user-specific model building process. Since all the computation – inference, tracking, building models – is carried out locally at the edge device, our proposed system ensures user privacy.

Finally, we propose architectural support to build user-specific models on heterogeneous edge devices comprising general-purpose CPUs and edge ML accelerators by enabling pruning on accelerators designed to support just the inference. We re-purpose Edge TPU, which computes inference in int8 precision, to also support the backward pass of the pruning phase in block floating point (BFP16) precision. We show that, by using bottom-up pruning and BFP16 precision, for the Resnet-50 model, we can reduce the model size by 2.6× and have accuracy comparable ($\leq$1% accuracy drop) to the original model while speeding up inference by 1.5×. Furthermore, for the Inception-V3 model, we can reduce the model size by 2.2× and have accuracy comparable ($\leq$1% accuracy drop) to the original model while speeding up inference by 2.25×. Moreover, the bottom-up pruning technique gives a pruning/training throughput of 10 images/sec and 7.54 images/sec for ResNet-50 and Inception-V3, respectively, on the re-purposed Edge TPU.

4.1 Background & Motivation

Machine learning (ML) has revolutionized technology in the past decade. It offers a wide range of applications, *e.g.*, computer vision [203], video recognition [202], and autonomous driving [158]. Machine learning is also used to design intelligent communication systems [116] that analyze complex scenarios in communication systems and make optimal predictions to obtain high Quality of service (QoS). For example, prior works [104, 194, 135] use a deep learning-based approach for MIMO detection. Furthermore, prior works [224, 161] have proposed deep learning solutions for the complex task of channel estimation. As a result of its increasing popularity, researchers have studied ML extensively for various computing platforms, including CPU [86, 230], GPU [230, 113, 218], and FPGA [84, 99, 193]. Its memory and compute-intensive nature have also led to the development of specialized architectures, including TPU [126], NPU [20], and several ML accelerators [209, 181, 68]. Recently, machine learning has emerged as a leading technique for improving the ways humans interact with machines. A few examples are voice recognition by IoT devices like Alexa/ Google Home, face recognition by smart cameras, and recommendation systems [174] for online shopping, personalized news feeds, and many more. Furthermore, many frequently used smartphone applications, like Facebook, Gallery/Photos, Instagram, Netflix, and so on, rely heavily on machine learning.

Despite its ubiquitous presence, machine learning is still one of the most latency-sensitive and energy-intensive applications for small, resource-constrained IoT/edge devices. IoT or edge devices are usually powered by tiny ARM cores or microcontrollers, which work along with a few specialized compute IP blocks or accelerators. Due to the limited compute capacity of edge devices, there is a wide gap between the performance of ML applications on edge platforms and server or desktop platforms. For instance, as shown in Figure 4.1, there is a wide gap between the execution time of large and accurate cloud ML models, like the Inception-V3 and Resnet-50, and small, but less accurate, edge-device friendly ML models, like Mobilenet and Shufflenet. Apart from performance, the compute and memory-intensive nature of ML applications also make them energy intensive, thereby reducing the battery life of edge platforms. Unlike servers or desktops that are always plugged into a power source, lithium-ion batteries power edge platforms. The small form factor of edge devices limits battery size and charge capacity. Furthermore, the portable and seldom remote nature of edge devices precludes their frequent charging. Hence, ideally, edge devices should be able to compute high-precision machine learning cloud models at the speed of edge models and consume minimal energy.

Today, to compute these large and accurate models, edge devices follow the common practice of offloading incoming machine learning requests to the cloud/back-end server. But such back-and-forth communication with the cloud raises additional issues. First, communicating to cloud servers requires fast and reliable internet connectivity, which can be a constraint in remote places. Second, transferring to the cloud leads to additional transmission latency and energy, which can impact overall performance and energy efficiency. Moreover, depending on the network traffic and available bandwidth, the transmission latency can result in a violation of the tight latency requirements of many popular machine learning-based applications. Third, and most important, offloading to a back-end server requires users to share their personal data with a back-end commodity server, leading to privacy and data-breach concerns. With the increasing frequency of cyber-attacks, sharing user data about every single activity may lead to harmful implications. For example, user data can be exploited to study the habits or daily routines of users, which can then be manipulated by malicious parties. All of the above concerns make it challenging to offload computation to the cloud reliably.

In order to address the above concerns, emerging techniques move computation closer to the edge/user device by computing either on edge *servers* or on edge *devices*. For example, [51] proposes a distributed solution based on game theory techniques to optimally offload partial computation to the multi-access edge servers in a risk-aware fashion. While an edge server-based distributed solution, like the above work, reduces the overhead and risk of cloud computing, computing entirely on edge devices completely eliminates those concerns. Hence, there have been significant efforts in pushing machine learning to edge devices [241, 61, 204]. One such emerging technique is Federated learning [60, 142], which endorses computation of all machine learning-

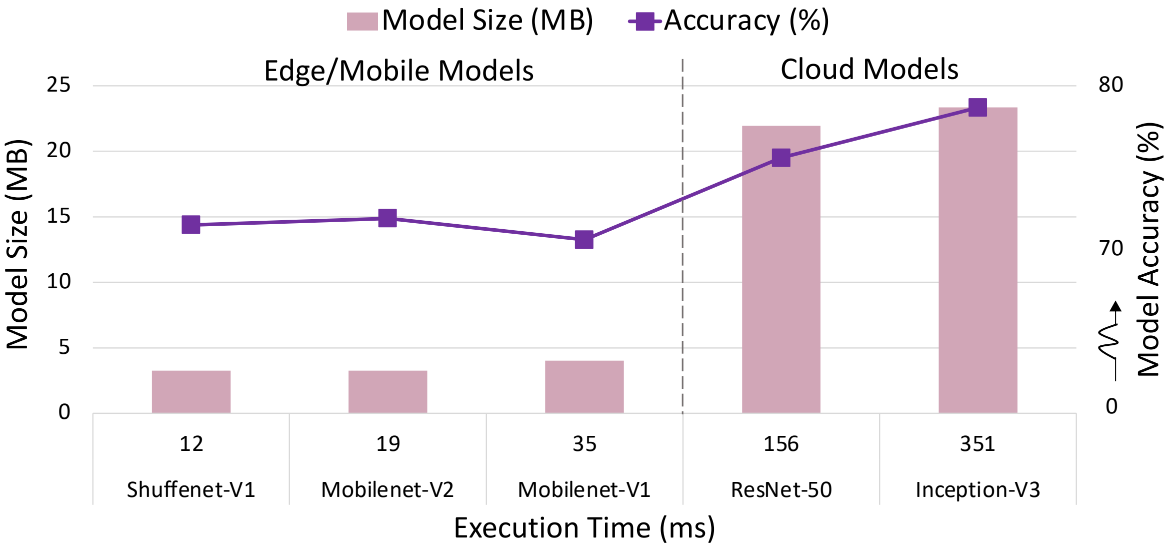


Figure 4.1: Comparison of accuracy, model size, and execution time across various edge and cloud ML models for ImageNet dataset.

related operations (ML inference and ML training) locally at the user device to eliminate privacy concerns related to sending user data back to the cloud. It trains the model at the edge and shares model updates (instead of raw data) with the back-end cloud. Federated learning enables privacy-preserving continuous training across many users but builds a *generic* model.

Another closely related work is transfer learning [228], a technique to learn models for new or smaller domains from already available trained models. It utilizes the top layers as it is from available trained models for the new domain and fine-tunes the remaining layer for the new dataset. Fixynn [219] is a transfer learning-based approach that builds multiple models for different domains/datasets via transfer learning by keeping the feature extraction layers constant and learning only the remaining layers. Our work is inspired by these efforts to keep all the ML computations local to the user device and build new models via transfer learning. We leverage the user interaction with the device to learn user preferences without sharing any data with other devices or the cloud. We then use this knowledge to make ML lightweight and more amenable to edge devices.

To address these challenges, in this work, we present MyML, a hardware-software solution that makes computationally intensive and accurate machine learning feasible at edge devices.

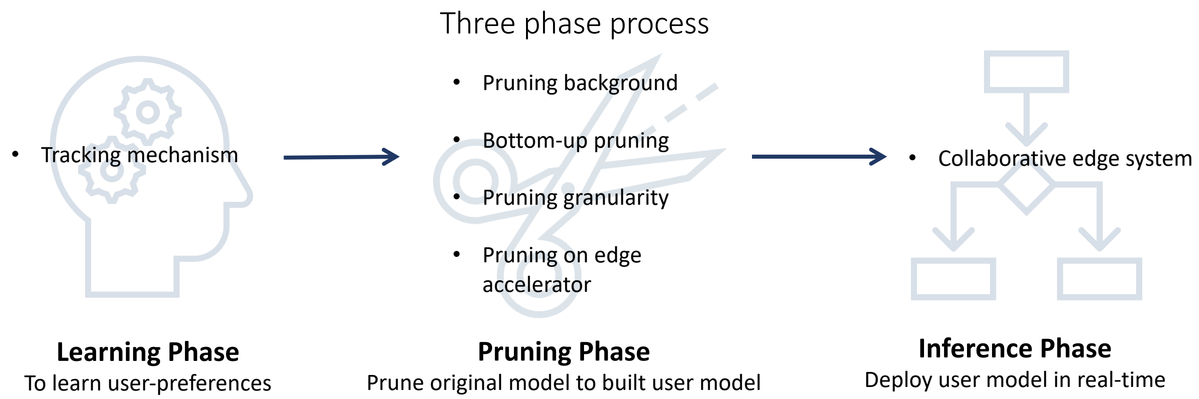

Three phase process
Tracking mechanism
Pruning background
Bottom-up pruning
Pruning granularity
Pruning on edge accelerator
Collaborative edge system
Learning Phase
To learn user-preferences
Pruning Phase
Prune original model to built user model
Inference Phase
Deploy user model in real-time

Figure 4.2: Our three-phase end-to-end process to learn user preferences, build user-specific model, and deploy it in real-time.

4.2 MyML Overview

Usually, machine learning models are built to serve numerous users with diverse choices and preferences; however, individual users have limited preferences. In this work, we explore the possibility of creating small, user-specific models according to user preferences rather than resorting to a larger generalized model for all users. We create such user-specific models based on the transfer learning method and avail user classes as a dimension to prune big ML models.

4.3 Building the User Model

As also shown in Figure 4.2, we employ a three-phase process to create and run a user-specific model as follows:

Learning Phase: In this phase, we use a *tracking mechanism* to learn user preferences based on the output of the original model. The tracking mechanism identifies the most frequent categories/classes in the first batch of input, termed as learning window, as user classes. The number of user classes depends on the number of categories with which the user frequently interacts. If any category contributes to more than x% of total inputs in the learning window, we consider it a user class. Our experiments have a tunable learning window of 50-100 images with an adjustable value of x. Therefore, for x of 15%, each category must appear at least 7-8 times among the total of 50 input trials. We also show a sensitivity study for increasing the number of user classes.

We have a naive learning phase, where we mark any category appearing more than x% (tunable parameter) in the learning window as a user class. We assume the non-frequent classes are one-time outliers. This approach can be modified to include any category encountered during the learning phase. Another possible approach is to utilize our thresholding mechanism from the collaborative edge system, discussed in a later section, to send the outliers back to the cloud server.

Pruning Phase: Pruning is defined as re-training of the current model with pruned weight set to zero. During the pruning phase, we use the incoming inputs to prune and re-train the original model to create a user-specific model for user classes learned during the first phase. We prune a block of layers for one epoch to optimize for training cycles. We use the output of the original model as ground truth for pruning. The goal is to build a user-specific model that has accuracy within 1% relative to the original model. Note that incoming inputs that do not belong to the classes identified during the learning phase are discarded and not used for pruning.

Inference Phase: After the pruning phase is complete and all layers are pruned, we switch the current working model at the edge device to the newly generated user-specific model. The pruning phase is a one-time process. Once the user-specific model is ready, inference can run efficiently without compromising accuracy until user data deviates from the current learned user classes. We discuss the process applied when user data begins to deviate from the learned preferences in Section 4.3.3.

4.3.1 Pruning Background

Building a user-specific model is a one-time yet expensive process. It requires pruning and weight updates of the original model for learned user classes, which are time and resource-consuming processes. Pruning is essentially re-training of the model with pruned weight set to zero value. It consists of two passes – forward pass and a backward pass. The forward pass is computationally the same as inference and is used to calculate the error between the output prediction of the current pruned model and ground truth. This final error, along with output activation for each layer, is utilized during the backward pass. The backward pass itself has two steps – error propagation and weight update. In error propagation, the final error calculated during the forward pass is propagated back to individual layers to get error contributions from each layer. This per-layer error, along with per-layer output activation stored during the forward pass, is used to find weight updates for each layer. The above-explained pruning mechanism involves compute-intensive and memory-intensive 2D and 3D convolutions, which makes it heavy for resource-constrained edge platforms.

The goal of this work is to build user-specific models locally at edge devices in order to preserve user privacy. To achieve this goal, we strive to make the complete pruning process lightweight by making the forward and backward pass less intensive. The first step is to use layer-wise pruning, where we prune/re-train one layer at one point in time while keeping the rest of the layers constant. The intuition behind layer-wise pruning is to share compute for the constant layers between the inference pass of the original model, currently serving incoming user requests, and the forward pass of the pruning model. Within layer-wise pruning, we explore two flavors of pruning – top-bottom pruning and bottom-up pruning.

In top-bottom layer-wise pruning, we start pruning from the top or first layers and iteratively move down to prune the bottom layers. In this approach, when, say, layer 1 is being pruned, the remainder of the bottom layers will be the same and can be shared with the original model. However, since the original model and the pruning model start diverging at the pruned layers, there will be two sets of incoming inputs to the shared layers – one input coming from the unpruned/original model top layers and the other input coming from pruned model top layers. We explore the possibility of using the difference (or delta) between the two sets of inputs to reduce compute. If there are a large number of zeros in the delta/difference, there can be a significant amount of compute sharing between the original model and the pruning model. However, contrary to our expectations, we did not find a significant fraction of zeroes in the delta. Hence, we did not pursue the top-bottom pruning approach further.

4.3.2 Bottom-up Pruning

The bottom-up pruning is inspired by the transfer learning [228] technique, where to learn new

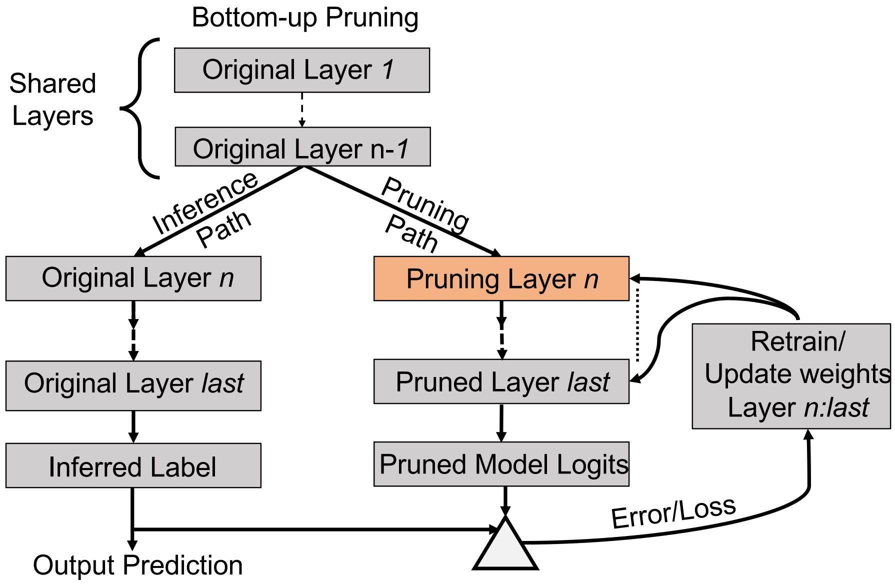


Figure 4.3: Bottom-up pruning shares compute for inference and pruning path until layer $n-1$. It diverges at the current pruning layer n and updates weights for layers n to $last$.

models in a different domain, the knowledge is transferred from an already learned model. The top layers of the available model are transferred as is to the new model, and the bottom layers, specific to the new domain, are trained from scratch. Deriving from this insight, in this work, we propose bottom-up layer-wise pruning. Here, instead of learning the bottom layers from scratch, we start pruning from the bottom-most layer and move up to prune the top layers in an iterative fashion. When a layer is being pruned, all the layers above it are kept the same as the original model layers and can share compute with it. Since the original model and pruning model start diverging at the current pruning layer; there is no input mismatch until this pruning layer. This property makes it easier to share compute between the original model and the pruning model. Therefore, in this work, we opt for the hardware-friendly, bottom-up pruning technique to reduce computation related to building the user-specific model. The bottom-up pruning mechanism reduces the computation required for the forward and backward pass of the pruning process, as discussed in detail below.

4.3.2.1 Compute Sharing between Inference and Forward Pass:

Since MyML carries out all the computation locally at the user device, it presents a unique opportunity of reusing inference computation for the forward pass of the pruning process. In bottom-up pruning, we perform layer-wise pruning starting from the last layer, as shown in Figure 4.3. While layer n is being pruned, all layers up to layer $n - 1$ are frozen and identical to the original model. Hence, both inference and pruning paths can share the computation carried out until layer $n - 1$. Starting from layer n, the inference path (to the left) will continue processing the original model to predict user output, while the pruning path (to the right) computes the remaining portion of the forward pass of the pruning process separately. Thus, only the current pruning layer n, and the layers below it, require separate processing during the forward pass. The pruning path calculates the error/loss based on logits (output of the last fully connected layer) produced by the pruned model and inference from the original model, which acts as the ground truth. This error is then used to compute weight updates for layers n to $last$. Note that we only have inference results from the original model available in real-time; hence, we use it as ground truth for the pruning phase. This methodology still achieves an accuracy within 1% of the original model for a dataset belonging to the user classes only.

4.3.2.2 Reduced Backward Pass:

The backward pass of pruning is a combination of 3D and 2D convolution for error-propagation and weight updates, respectively [164]. Bottom-up pruning provides the opportunity to reduce the number of weight updates and computations for the backward pass. The proposed scheme freezes all the layers until layer $n-1$. Thus, we do not require weight updates for layers 1 to $n-1$. Weight updates are needed only for the current pruning layer n and beyond, reducing overall pruning time. Furthermore, since we are progressively pruning layer by layer, when pruning layer n, all the layers $> n$ will have already been pruned and left with just unpruned channels/weights. Thus, the layers $n + 1$ to layer $last$ require weight updates for fewer channels, further reducing computation time as we progressively prune layers in a bottom-up fashion. A significant advantage of the bottom-up pruning technique is that we are able to keep to the original model (and its accuracy benefits) while reusing the compute for top layers for the pruning process. Once all the layers up to the first layer are pruned, we replace the original model with the user model.

Bottom-up pruning complexity: The proposed framework has an additional pruning path to train the user-specific model. This comes with the added complexity of forward pass and backward pass. As discussed before, the backward pass of a convolution layer is comprised of 3D convolution for error propagation & 2D convolution for weight update, both of which are based on matrix-matrix multiplication kernel. Furthermore, the forward pass, which is computationally similar to inference, consists of matrix-matrix multiplication-based 3D convolution for each layer. The matrix-matrix multiplication has a worst-case computational complexity of O(n^3). The pruning path adds 3 matrix multiplication kernels corresponding to forward and backward pass to update weights of each convolution layer; thus, it increases the complexity linearly by 3 times, *i.e.*, 3*O(n^3). Equations 4.1 and 4.2 show the computational complexity of pruning and inference paths. For equation 4.1, the number of updates varies for each layer, *i.e.*, the bottom layers, which are pruned first, will have more number of updates and will decrease as we move to top layers, which are pruned towards the end.

$$Total\_complexity\_pruning\_path = \sum_{i=bottom}^{top} num\_updates\_lyr\_i * (3 * O(n^3)) \tag{4.1}$$

$$Total\_complexity\_inference\_path = total\_num\_layer * (O(n^3)) \tag{4.2}$$

4.3.3 Collaborative Edge System

Once the user-specific model is constructed during the pruning phase, using bottom-up pruning and the appropriate pruning granularity, we replace the original model with the user model and enter the inference phase. However, the user can still switch or change preferences with time. If the user-specific model trained for previous user preferences is utilized for new preferences/choices, it will lead to severe accuracy degradation. Hence, there is a need for a mechanism to detect a switch in user preferences. Therefore, for the inference phase, we develop a collaborative edge system to

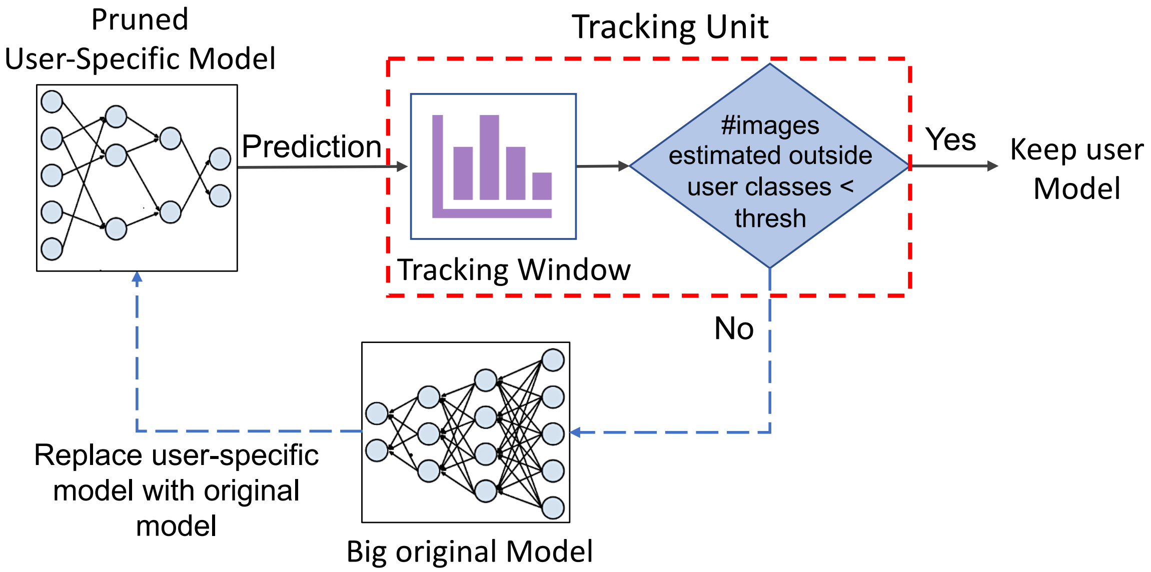


Figure 4.4: Collaborative edge system with a tracking unit that checks for divergence in user preferences by counting the number of predictions belonging to user classes.

find a deviation in user preferences with a tracking unit, as shown in Figure 4.4. In this adaptive system, we use the newly created user-specific model to classify incoming inputs and a tracking unit to count both the number of inputs classified within user classes and the number of inputs classified outside of user classes. If the tracking unit classifies the majority of inputs as outside of user classes, we conclude that user preferences have changed and restart the user model building process. Since the user-specific model is biased to predict one of the user classes, we cannot directly use the predictions to determine if the incoming input is within or outside of user classes. Instead, we use the probability of the predicted user class and entropy of the probability distribution over all the user classes as the metric to tag the input to a user class or out-of-user class. The output probability is high, and entropy is low for inputs belonging to the user classes. In contrast, the output probability is low, and entropy is high (implying the model has low confidence) for inputs belonging to out-of-user classes. Any input with an output probability less than a threshold and entropy higher than an empirically determined threshold is marked as belonging to an out-of-user class. For a running window of 50 images, if more than 70% of images are estimated to be outside learned user classes, we infer that user preferences have changed and discard the current user-specific model. We then restore the original model and restart the learning phase of the user model creation process to identify new user preferences. We assume the large original unpruned model to be stored in flash (or DRAM) and fetched from there on being summoned by the CPU for the classification task.

The threshold selection for output probability and entropy is guided by the statistical definition of the parameters. For the output prediction probability, the threshold should be at least 50% to claim that the input was confidently predicted. Thus, to make our adaptive system more robust, we chose a slightly higher value of 60% (or 0.6). Furthermore, for the entropy threshold, we first determined the maximum entropy. The maximum entropy of the prediction probability distribution is when all the classes have an equal probability of prediction. For a model built for 5 user classes, the equal probability is 1/5 (0.2), which upon plugging into the entropy equation, ($\sum$ -plog(p)), gives the max entropy value of 2.32. To further reinforce model confidence and robustness for our collaborative system, we set the entropy threshold to 1.5, which is less than half of the maximum value.

4.4 Pruning Granularity

Our pruning techniques are guided by the underlying hardware computation granularity. For example, a CPU with a SIMD width of 16 slots can compute and prune at the finest granularity of 16 continuous operations in parallel. If a few of the operations, say slots 8 and 10, are zeroed during pruning, we cannot skip the two operations and get performance benefits since the CPU computes those 16 slots together in parallel. The CPU can skip computations if all 16 operations are zeroed out during pruning. Thus, to utilize the parallelism provided by the compute engine, for example, the SIMD width of the CPU, we have to map only non-zero contiguous operations to each block of compute. Hence, in this work, to create pruned user-specific models, we explore two kinds of structured pruning techniques – symmetric channel pruning and asymmetric channel pruning. Since channel pruning zeroes out all continuous weights in a pruned channel, we can skip com-

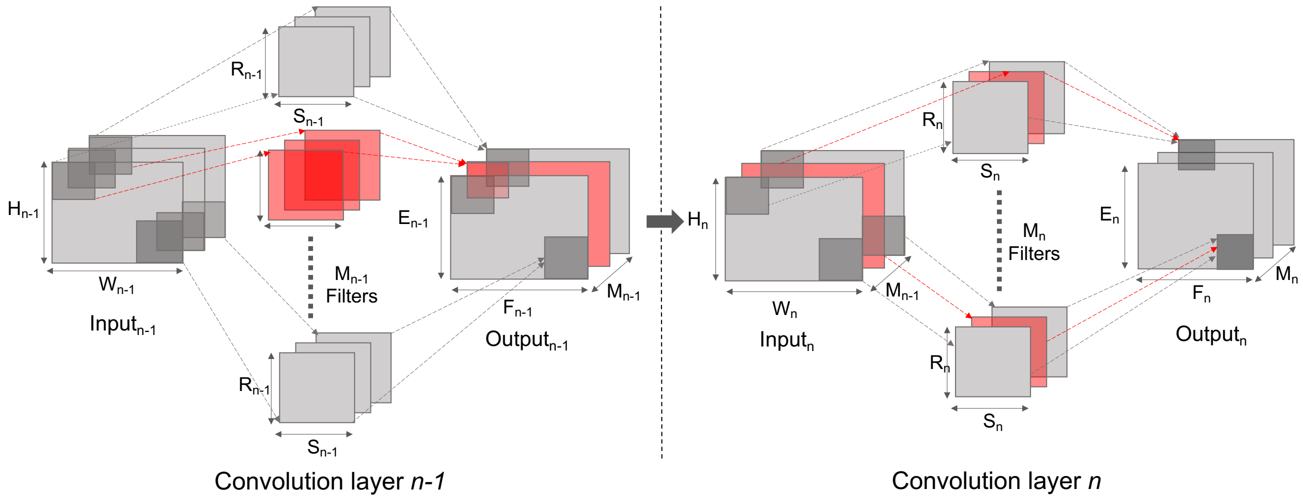


Figure 4.5: Symmetric Channel Pruning: As a result of pruning the same channel IDs across all filters in layer n, the corresponding (red) filter in layer $n-1$ is pruned completely. All channels and connections shown in red are pruned.

putations for all weights of the channel altogether to improve performance and energy efficiency. Furthermore, prior work has shown that classic sparse formats can lead to performance loss for pruned weight matrices [230]. Our choice of an entire channel (2D filter) as a pruning granularity allows us to retain the dense format for inference computation, even with pruned weight matrices.

4.4.1 Symmetric Pruning

In symmetric channel pruning, the same channel IDs are pruned across all the 3D filters in a given layer, which is a more constrained approach illustrated in Figure 4.5. Channel IDs with the lowest L2 norm values, summed across all the filters, are pruned (shown in red) in layer n. This step further leads to the pruning of filters in the previous layer $n-1$, corresponding to channel IDs pruned in the current layer n. Symmetric pruning changes the existing dense convolution layers structure to another dense convolution layer structure; hence it *does not require* any bookkeeping mechanism for convolution layers to store which channels were pruned. The removal of filters in layer $n-1$ removes the corresponding output channel and the input channel for the subsequent layer n. Therefore, in layer n, only input channels corresponding to unpruned filter channels are

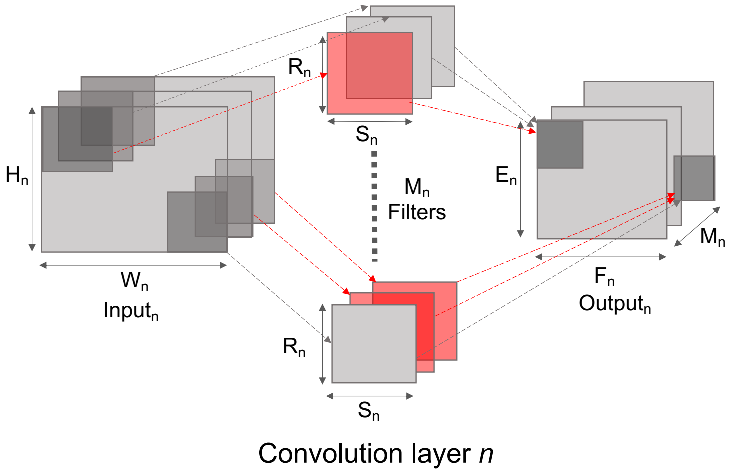


Figure 4.6: Asymmetric Channel Pruning: Channels are pruned independently with no restrictions. More channels are pruned with this approach because of its flexible nature. All channels and connections shown in red are pruned.

present. Thus, we do not need to store any extra channel information for the convolution operation to map input channels correctly to the remaining filter channels. Note that for identity mapping in residual networks like ResNet, we need to store unpruned channel indexes to gather unpruned channels and drop the pruned channels in the identity path for the final addition operation of the identity branch and the parallel running convolution branch.

4.4.2 Asymmetric Pruning

In asymmetric pruning, we prune individual channel IDs with the least L2 norm values, across all the filters, with no restriction to choose the same channel IDs across filters, as shown in Figure 4.6. This makes the approach more flexible and boosts the potential for increasing the pruning rate. Asymmetric pruning maintains the current dense model structure with only unpruned channel weights but requires a bookkeeping mechanism to store only unpruned channels in a dense format.

Bookkeeping:

The purpose of the bookkeeping mechanism is to store only non-zero channels and map them correctly to their corresponding input channels. Each filter of the convolution layer requires different input channels. For instance, in Figure 4.6, the first channel for the top filter is pruned; thus, only the last two channels need their corresponding input channels. Similarly, the last two channels are pruned for the bottom filter; thus, it requires only the first input channel for computation. Due to this asymmetric behavior, we store all the input channels in dense format. However, we need to map non-zero channels in each filter to their corresponding input channels. We achieve this through the bookkeeping mechanism, where we only store non-zero filter/weight channels in a contiguous manner, along with some meta-data. To map weight/filters channels to their corresponding input channels, we store a channel mask offset, the number of non-zero channels [nnz] per filter, and the difference pointer {diff} between consecutive non-zero channels to point to the correct input channel.

Convolution is a matrix multiplication operation between flattened 2D input and flattened 2D weights. Each row of the input activation matrix is one input window from the Image-to-column (Im2Col) operation, which is multiplied by different filters represented by columns of the weight matrix. We show a small working example in Figure 4.7. The input pointer starts with channel mask offset (in orange) to compute the first channel and then uses the stored difference pointer {diff} (in blue) to move to the next non-zero channel. For instance, to skip pruned channels and move to the next non-zero channel, we store a diff pointer value of {+3}. We simultaneously keep a counter of the number of non-zero channels computed/visited. When the counter is equal to the store number of non-zero channels (nnz) value for the filter, it indicates that no more non-zero channels are present in the filter. We use this as an indicator to perform the accumulation or aggregation operation across all the filter channels. In Figure 4.7, for filter F1, we do aggregation when the counter becomes equal to nnz value of 2 after computing the two non-zero channels of the filter. To move to the first non-zero channel for the next filter, F2, we reset the counter and accumulator and then simply use the diff pointer {-2} to move to the correct location. This approach incurs a small overhead for storing meta-data of the mask offset, difference pointer, and the number of non-zero channels. However, this overhead corresponds to merely 24KB per layer for our pruning rates.

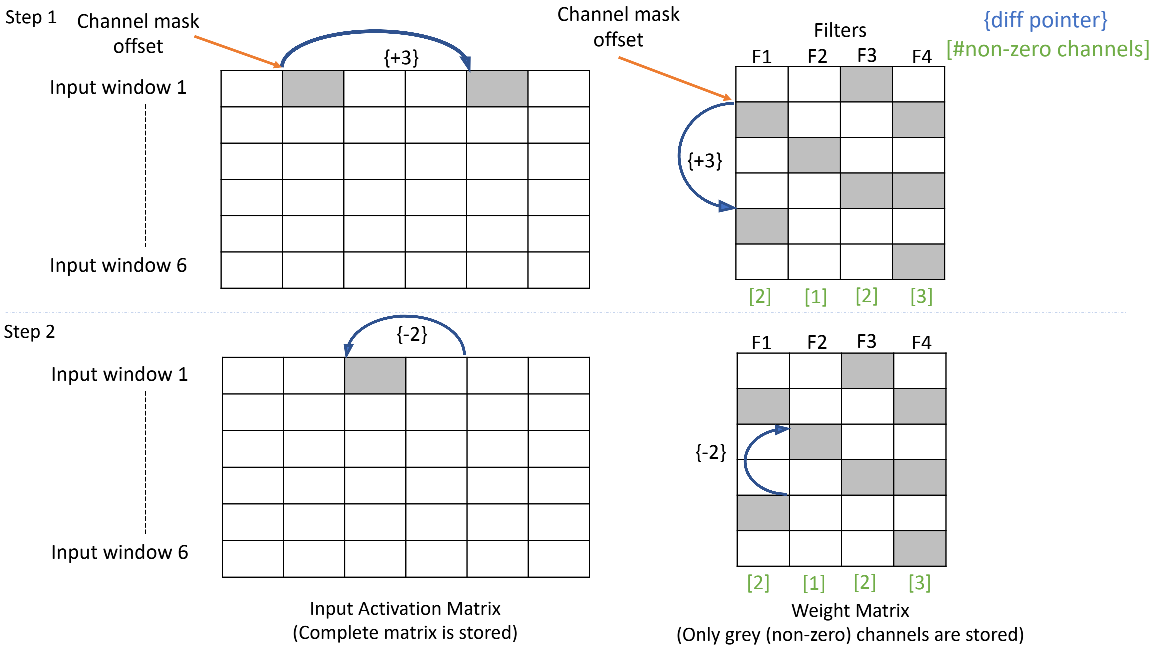


Figure 4.7: Bookkeeping mechanism with channel mask offset, difference point {diff}, and the number of non-zero channels [nnz].

4.4.3 Which pruning granularity to use

The choice of pruning granularity depends on the user platform on which the machine learning inference will be computed. Mobile platforms with only multi-core CPUs or mobile apps that prefer to run machine learning on CPUs [221] should opt for asymmetric pruning because CPUs can control and compute at a finer granularity, such as SIMD width. CPUs can efficiently handle the bookkeeping mechanism required to support asymmetric pruning and thus benefit from high pruning rates. However, edge accelerators that map machine learning computation to dense 2D systolic arrays do not have the capability to control or skip computations at fine granularity. Thus, such systolic-array-based edge accelerators can take advantage of symmetric pruning, which does not require any bookkeeping, while still availing themselves of the benefits of pruning.

4.5 Pruning on Edge Accelerator

Edge accelerators or neural engines for machine learning-related tasks have become an integral part of most of the state-of-the-art mobile SoC platforms [20, 31]. In such a heterogeneous CPU-edge

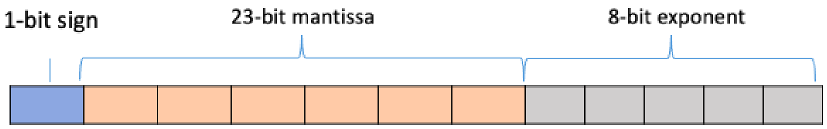


Figure 4.8: FP32 precision: 1-bit sign, 23-bit mantissa, & 8-bit exponent for each element.

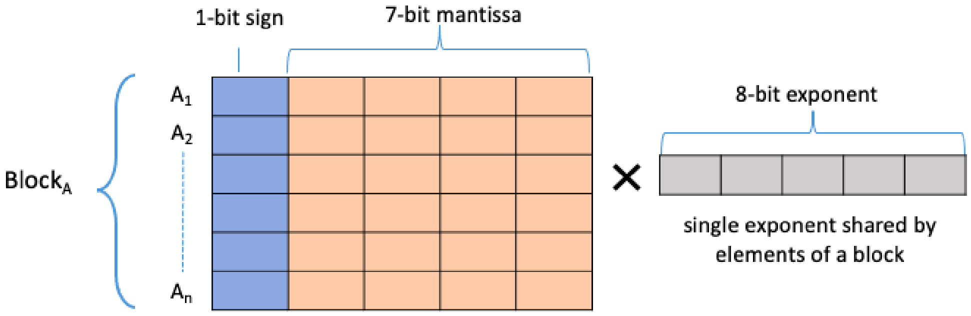


Figure 4.9: Block floating point (BFP16) precision: 1-bit sign and 7-bit mantissa for each element. 8-bit shared exponent across all the elements in a block.

accelerator system, general-purpose CPU cores offload the incoming ML inference requests to neural engines for faster and more energy-efficient computations. Unlike multi-core CPUs, where the parallelism is limited by supported SIMD width (in order of 10s), edge accelerators provide a high level of parallelism with hundreds of compute units working in parallel to produce the final output. Therefore, in MyML, we also propose architectural techniques to enable the pruning phase on a heterogeneous CPU-edge accelerator system. In this work, we use an accelerator modeled after Google's Edge TPU [12].

Edge TPU is a systolic array-based ML inference accelerator that supports processing elements (PEs) with an 8-bit multiply-accumulate (MAC) operation. It is used to compute the General Matrix Multiplication (GEMM) kernel that forms the backbone of ML inference and training. For an $N \times N$ systolic array, each column computes N MAC operations in parallel to produce an output activation. Thus, in one cycle, N output activations (one from each column) are computed in parallel. Moreover, EdgeTPU has streamlined data-flow where the read/write latency of input/output is hidden by MAC computation with only one-time weight-loading latency visible. Hence, Edge TPU outperforms a multi-core SoC with ARM ISA cores [13], which does not have pipelining or overlapping and also has general-purpose instruction processing overheads. Edge accelerators are designed to support only inference in int8 precision due to power and area constraints. However, the backward pass of the training process needs floating-point precision to compute weight updates. Supporting a dedicated accelerator for floating-point is expensive in terms of area/power. It is also not justified for short pruning periods since the majority of the time is spent on inferences. Hence, to make our solution practical for edge platforms with an inference accelerator, we re-purpose an int8 edge accelerator to enable the higher precision needed for pruning. The re-purposing technique proposed in this work can have a broader scope and be used for Cloud TPU or other systolic array-based ML compute engines.

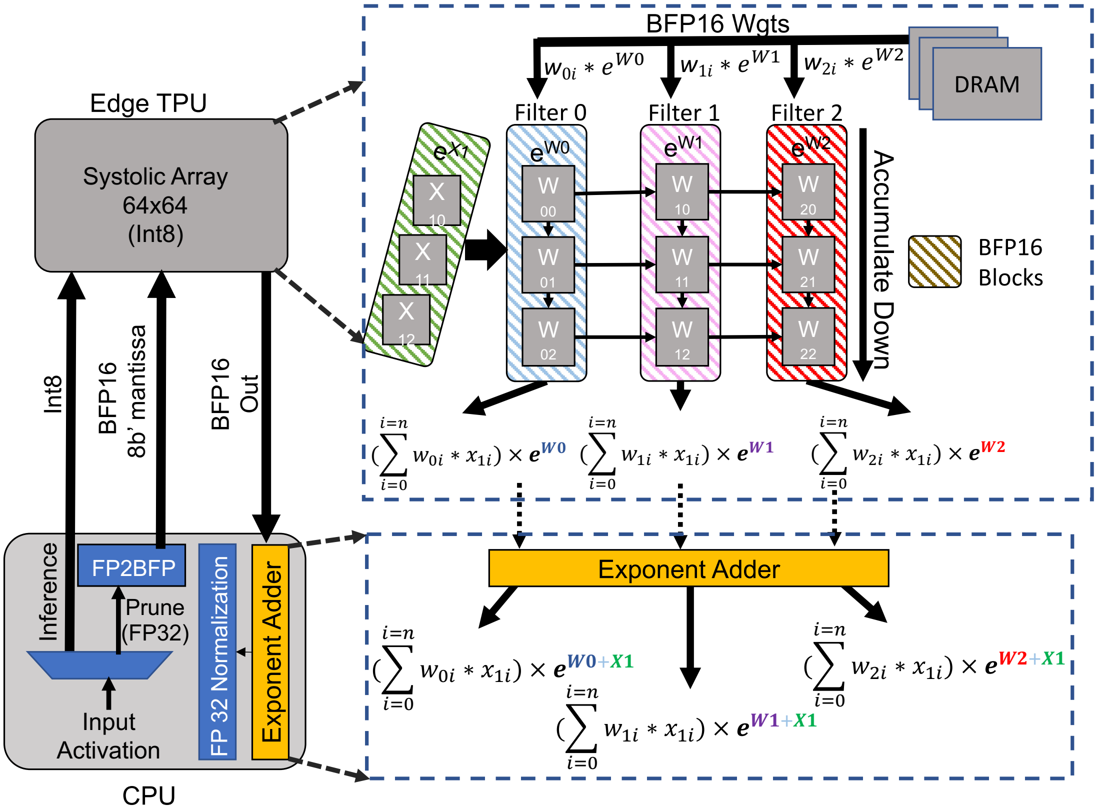
4.5.1 Repurposed Edge TPU

In order to re-purpose the int8 precision edge accelerator for pruning, we want to leverage the int8 computation done at Edge TPU and append it with some lightweight computation done at the CPU. Therefore, we propose using Block Floating Point (BFP) [84] as an alternative to the FP32 floating-point format to compute the backward pass on Edge TPU. For FP32 representation, each element has a dedicated sign, mantissa, and exponent bits, as shown in Figure 4.8. In contrast, for BFP representation, each block/vector shares the same exponent across all the elements, while each element in a block has an individual mantissa, as shown in Figure 4.9. The dot product of any two BFP blocks is shown in Equation 4.4 where m_a, m_b are mantissa bits and e^{a}, e^{b} are exponents for the Block_a and Block_b. The dot product of the two blocks is the dot product of the mantissa bits, while the exponents can simply be added to get the exponent for the final dot product. In MyML, Edge TPU is used to compute the dot product of the mantissa bits of weights and input activation, while the host CPU appends the BFP output from the Edge TPU with the sum of the exponent bits. Thus, the BFP format maps well to Edge TPU because the computation related to shared exponents can be processed at the CPU, and the computation for mantissa bits can be completed separately at Edge TPU. We use the BFP16 floating-point format, which has an 8-bit signed mantissa and 8-bit exponent. The 8-bit signed mantissa can be mapped directly on an Edge

TPU architecture, supporting 8-bit fixed point MAC PEs.

$$Block_a = \{m_{a_1}, ..m_{a_n}\} \times e^a \quad Block_b = \{m_{b_1}, ..m_{b_n}\} \times e^b \tag{4.3}$$

$$Block_a \cdot Block_b = (\sum_{i=1}^{n} m_{a_i} m_{b_i}) \times e^{a+b} \tag{4.4}$$



BFP16 Wgts
DRAM
$w_{0i} * e^{W0}$
$w_{1i} * e^{W1}$
$w_{2i} * e^{W2}$
Filter 0
Filter 1
Filter 2
e^{X_1}
X 10
X 11
X 12
e^{W0}
e^{W1}
e^{W2}
W 00
W 01
W 02
W 10
W 11
W 12
W 20
W 21
W 22
Accumulate Down
BFP16 Blocks
Edge TPU
Systolic Array 64x64 (Int8)
Int8
BFP16 8b' mantissa
BFP16 Out
Inference
FP2BFP
Prune (FP32)
Input Activation
FP 32 Normalization
Exponent Adder
CPU
$(\sum_{i=0}^{i=n} w_{0i} * x_{1i}) \times e^{W0}$
$(\sum_{i=0}^{i=n} w_{1i} * x_{1i}) \times e^{W1}$
$(\sum_{i=0}^{i=n} w_{2i} * x_{1i}) \times e^{W2}$
Exponent Adder
$(\sum_{i=0}^{i=n} w_{0i} * x_{1i}) \times e^{W0+X1}$
$(\sum_{i=0}^{i=n} w_{1i} * x_{1i}) \times e^{W1+X1}$
$(\sum_{i=0}^{i=n} w_{2i} * x_{1i}) \times e^{W2+X1}$

Figure 4.10: Re-purposed Edge TPU for training.

Figure 4.10 shows the complete block diagram for the re-purposed Edge TPU. Each column of the systolic array is mapped to the individual filters of a layer and represents one block sharing a common exponent. Similarly, each column of the input vector streaming into the systolic array is mapped to an individual input window and represents one block sharing one exponent. Before computation, BFP16 weights are loaded from DRAM into the PEs of the systolic array. This weight loading is a one-time process, which is followed by a long computation phase. Only 8-bit mantissas of loaded BFP16 weights are used during the computation phase. The 8-bits exponent flows through directly to the output. During the computing phase, 8-bit mantissas of input blocks are streamed into the 2D systolic array, and the output of each column of the systolic array is the dot product of filter weights and input window activation mapped to that column. In one cycle of the computing phase, Edge TPU completes N MAC operations in each of the N columns, resulting in $N \times N$ completed operations in one cycle. The output from the Edge TPU is piped out to the host CPU, which then appends the output with the sum of the exponent bits of input and weight block to deliver the exponent bits for the final output in BFP16 format. Furthermore, to accumulate the output for large filters spanned across multiple GEMM kernel calls, we use normalization to convert the BFP16 output from Edge TPU to FP32. The FP32 outputs can then be easily accumulated across multiple GEMM kernel calls to obtain the final output for the filters.

To support the CPU-repurposed Edge TPU system, we add three components to the host CPU. The first component is a floating point to block floating point converter (FP2BFP) that converts the input and intermediate activations in FP32 precision to BFP16 precision. The mantissa bits of the BFP16 inputs are sent to Edge TPU for further computation. The second component is the exponent adder, which adds the 8-bit exponents of input and weight BFP16 blocks to give the final exponent of the output block. The last component is for FP32 normalization to convert BFP16 output blocks to FP32 blocks. Thus, our system fully implements the conversion process between FP32 and BFP16 (and vice-versa) on the host CPU, as shown in Figure 4.10.

4.5.2 Conversion Error from FP32 to BFP16

Conversion from FP32 precision, which has 23-bit of mantissa and 8-bit of exponent (as shown in Figure 4.8), to BFP16, which has 8-bit of signed mantissa and 8-bit of exponent and wherein exponent is shared by all the elements within a block (as shown in Figure 4.9), leads to a deviation of current/working weight and activation values from original values. The error due to the conversion of FP32 values to BFP16 is because of the reduction in mantissa bits and the block size, *i.e.*, the number of elements in a block of BFP16. While mantissa bits must be set to 8 to match the underlying Edge TPU precision, the block size is flexible. Smaller block size reduces the chances of overflow or underflow and the divergence from original (FP32) values, thus, reducing the error.

However, FP32 to BFP16 conversion overhead depends on the number of BFP16 blocks to convert. It increases for small block sizes with more BFP16 blocks. Hence, it is a trade-off between the sizes of each block and the number of blocks. Moreover, the minimum block size is limited by the dimension of the systolic array present at Edge TPU. The small dimension of Edge TPU (*i.e.*, 64) constrains the block size and reduces the chances of overflow or underflow, resulting in smaller errors. Thus, the block size is set to 64 to match Edge TPU's 64x64 systolic PE block. One column/row of matrix multiply is divided into multiple blocks of size 64, *i.e.*, 64 elements in each block. Furthermore, even using a small block size of 64 and mantissa width of 8, we observe a significant drop in model accuracy with BFP16 precision. The original unpruned Inception-V3 model with FP32 precision has an accuracy of 79.2% for the user-specific dataset. This accuracy drops down to 76% when using BFP16 precision for the same unpruned original model. Thus, there is a significant, 3.2%, accuracy drop from the error generated due to conversion from FP32 to BFP16 precision. We regain this accuracy drop by re-training the model during the user-specific pruning process to achieve 79.2% accuracy.

4.6 Methodology

We evaluate MyML for the image classification task with the Inception-V3 [211] and Resnet-50 [105] models. We show results primarily for a user-dataset comprising five randomly chosen classes from the Imagenet dataset [77], representing user preference. We prune the model using TensorFlow's tf-slim framework to obtain pruning rates and measure accuracy. We also extend the TensorFlow framework to support block floating point (BFP16) precision by adding a floating point (FP32) to the BFP16 conversion module. This is a generalized module that can be configured for different block-size, mantissa bits, and exponent bits. For our experiments, we have set the block size to 64, exponent bits to 8 bits wide, and mantissa bit to 7 bits with one additional bit to represent the sign bit.

We use the XNNPACK [32] library for mobile CPU performance evaluation, which provides a

Cores	1xA76@2.84GHx, 3xA76@2.41GHz, 4xA55@1.78GHz
L1 cache	1x128KB, 3x128KB, 4x128KB
L2 cache	1x512KB, 3x256KB, 4x128KB
L3 cache	2MB
DRAM	LPDDR4 6GB@2133MHz, 34.1GB/s

Table 4.1: Architectural specifications for Snapdragon 855 Octa-core SoC representing mobile CPU.

SIMD implementation of 3D convolution using the ARM Neon ISA. We extend this library further to add SIMD support for 2D convolution. Using the GEMM implementation of XNNPACK[32], we can skip entire blocks, corresponding to pruned channels, for asymmetric pruning with the bookkeeping mechanism explained in Section 4.4. We measure execution time and energy consumption by executing these kernels on Samsung S10e mobile phone hosting Snapdragon 855 Octa-core mobile SoC with the complete architectural configuration listed in Table 4.1.

To evaluate the performance for Edge TPU, we use the SCALESim [192] simulator, which gives compute cycles for a given systolic array configuration and assumes a TPU operating frequency of 500MHz. Since SCALESim supports only 3D convolution, we extended it to support 2D convolution for the backward pass.

As discussed in Section 4.2, our learning window size is 50-100 images, and the minimum appearance frequency is $\sim$15% of window size for a class to be marked as a user class. We divided the total layers into four blocks and pruned one block at a time, starting from the bottom block, per our proposed bottom-up pruning technique. Each block was trained for 40 images per user class, which accrued to 200 images for the five-class user-dataset, accounting for a total of 1000 images for the pruning phase. Furthermore, to optimize for accuracy as well as training cycles at the mobile device, we trained each block for one epoch, with the option of training the last block for multiple epochs to improve model accuracy. We trained the last block for simply one additional epoch to build a robust user-specific model in our experiments.

4.7 Evaluation

We evaluate MyML on two distinct platforms – mobile CPU and Edge TPU – with various pruning configurations for Inception-V3 and ResNet-50 models. For mobile CPU, we compare the user-specific model pruned using asymmetric pruning with two baseline models: the original unpruned model with pruning type as none and the original model pruned using channel pruning (user-agnostic), which represents the prior user-agnostic pruning works. For TPU, we compare the user-specific model pruned using symmetric pruning with the original model. Note that Edge TPU is designed for dense GEMM matrix computation and cannot accrue the benefits of user-agnostic channel pruning; hence, we do not report any user-agnostic pruning configuration for TPU.

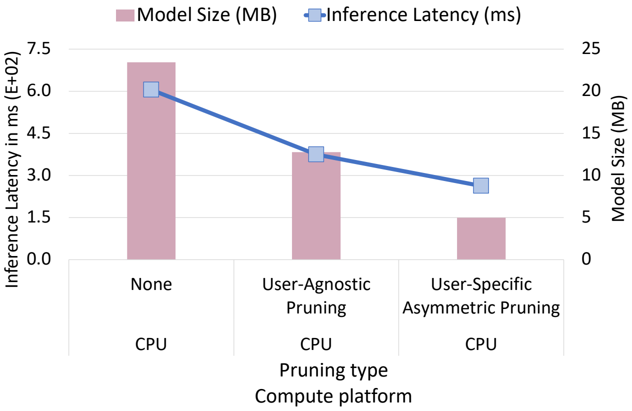


Figure 4.11: Inception-V3: Inference latency and model size for different pruning types on mobile CPU platform that supports Int8 precision for inference and FP32 for pruning.

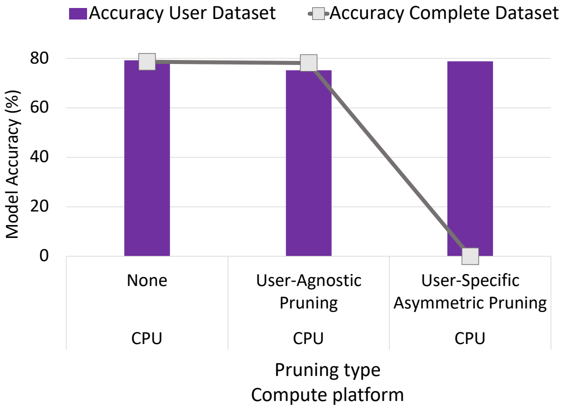


Figure 4.12: Inception-V3: Model accuracy for user-specific dataset and the complete Imagenet dataset for different pruning types on mobile CPU platform that supports Int8 precision for inference and FP32 for pruning.

4.7.1 Inception-V3

The inception model was first developed by Szegedy et al. [210]. It was an important milestone because it shifted the contemporary trend of building deeper models to wider models. Deeper models are more prone to over-fitting. Hence, instead of having one filter at one level, these models include multiple filters at one level to form a wider network. The Inception-V3 model [211] is an

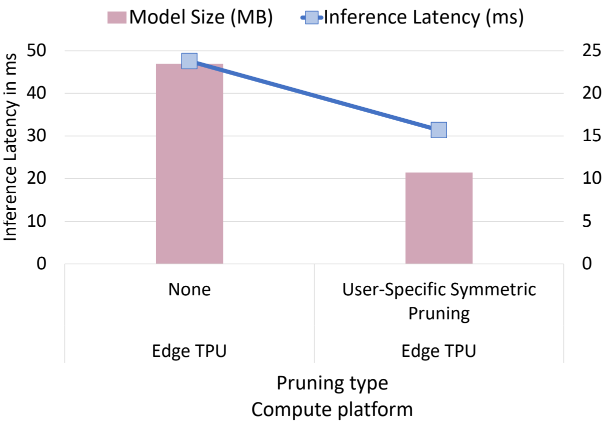


Figure 4.13: Inception-V3: Inference latency and model size for different pruning types on Edge TPU platform that supports Int8 precision for inference and BFP16 for pruning.

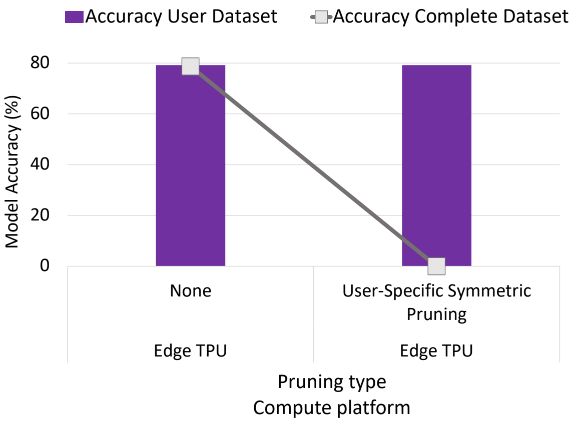


Figure 4.14: Inception-V3: Model accuracy for user-specific dataset and the complete Imagenet dataset for different pruning types on the TPU platform that supports Int8 precision for inference and BFP16 for pruning.

advanced version that reduces computational bottlenecks.

Inference Performance and Accuracy. As shown in Figure 4.11, we find that the user-specific model built using asymmetric pruning on the mobile CPU is 2.3× faster, corresponding to a 4.7× reduction in model size, as compared to the original model. Moreover, compared to the pruned user-agnostic model, the user-specific model provides a 1.4× speedup, along with a 2.5× reduction in model size. The newly built user-specific model has an accuracy of 78.8% (less than a 1% accuracy drop in user-dataset) compared to the original model with an accuracy of 79.2 %, as shown in Figure 4.12. The user-specific model has higher accuracy compared to the user-agnostic pruned model for the user-datatset because the user-specific model is pruned (and re-trained) only for user classes. On the other hand, the user-agnostic model is pruned to maintain combined average accuracy across all the 1000 classes of the complete Imagenet dataset. Henceforth, the accuracy on the complete dataset is maintained by the user-agnostic pruning, whereas the accuracy drops to $< 1\%$ (close to zero) for the user-specific model because the inputs belong to outside user classes. Thus, we can conclude that the user-specific model yields an accuracy comparable to the original model for inputs belonging to user classes but does not work for inputs outside user classes, reinforcing the correct behavior of user-specific models.

Platform	Pruning Type	Precision	Pruning Phase Duration
mobile CPU	User-Specific Asymmetric	FP32	390.34 (s)
Edge TPU	User-Specific Symmetric	BFP 16	132.6 (s)

Table 4.2: Inception-V3 pruning comparison between mobile CPU and repurposed Edge TPU.

For inference on Edge TPU, we observe that inference time and model size reduce by 2.25× and 2.2×, respectively, for the user-specific model built with symmetric pruning over the original model (as shown in Figure 4.13), while maintaining an accuracy of 79.2% over the user-dataset (as shown in Figure 4.14). Furthermore, similar to the mobile CPU platform, accuracy also drops to $< 1\%$ (close to zero) for the complete Imagenet dataset on the Edge TPU platform.

There are two factors that contribute to performance improvement in Edge TPU. The first is due to the reduction of model size because of channel pruning. The second is the reduction in the Image-to-column (Im2col) operation that is a part of the pre-processing step. Inputs can be piped out to the Edge TPU only once they are flattened out and converted to a 2D matrix to map to a 2D systolic array. This operation depends on the number of input channels of the convolution layer. Since we remove complete filters and corresponding output/input activation channels as part of symmetric pruning, we end up reducing Im2col operations as well. This leads to additional performance benefits over the GEMM operation reduction.

Pruning Performance: In Table 4.2, we report the duration of the pruning phase comprising 1,000 images. We find that the mobile CPU with asymmetric pruning can process 2.56 images/sec, which accumulates to a total time of 390s for the pruning phase. Edge TPU with symmetric pruning can process 7.54 images/sec, aggregating to 132s for the pruning phase. Our repurposed Edge TPU is able to reduce pruning time by $\approx 3\times$. We expect the pruning to be a one-time cost for long inference phases where user classes remain stable.

Energy: We also observe improvement in the energy efficiency of computing the models on our mobile device. The energy per inference reduces to 0.98J for the user-specific model, compared to 1.54J and 1.27J for the original and pruned user-agnostic model, respectively. This results in energy reductions of 54% and 27%, respectively, for the user-specific model compared to the pruned original model and the original unpruned model.

4.7.2 Resnet-50

ResNet-50 is a crucial machine learning model for image recognition/classification tasks and has been widely adopted by industry and academia. It is an integral part of the MLPerf's [166] AI inference and training benchmark suite for datacenter, developed in collaboration with academia, research labs, and industry, with reasonable accuracy of 75.6% with a 21.7 MB model size. It was the first network to introduce the concept of identity mapping [105], which made training easier and improved generalization. In this work, we include ResNet-50 in our experiments to demonstrate the benefits as well as the broad applicability of MyML. We generalize that the MyML technique can be applied to any deep neural network with convolution layers.

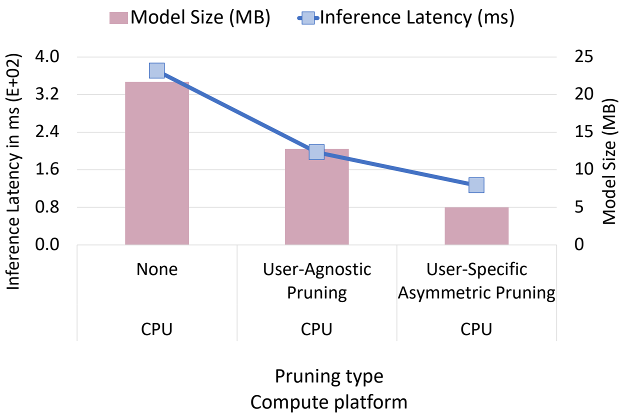


Figure 4.15: ResNet-50: Inference latency and model size for different pruning types on mobile CPU platform that supports Int8 precision for inference and FP32 for pruning.

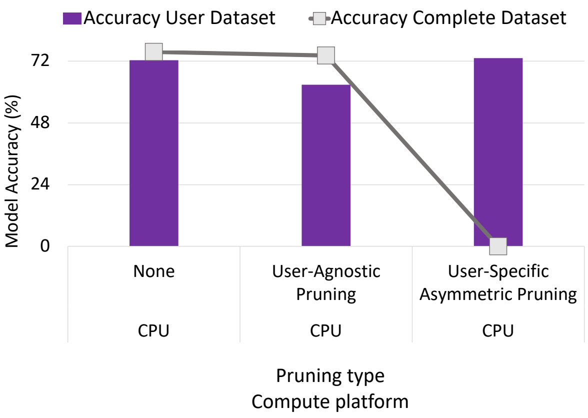


Figure 4.16: ResNet-50: Model accuracy for user-specific dataset and the complete Imagenet dataset for different pruning types on mobile CPU platform that supports Int8 precision for inference and FP32 for pruning.

Inference Performance and Accuracy. As shown in Figure 4.15, we find that the user-specific model built using asymmetric pruning on the mobile CPU is 2.93× faster, corresponding to a 4.3× reduction in model size, as compared to the original model. Moreover, compared to the user-agnostic model, the user-specific model provides a 1.55× speedup and a 2.5× reduction in model size. The newly built user-specific model has an accuracy of 73.2%, which is within a 1% accuracy margin, compared to the original model with 72.4% in user-dataset, as shown in Figure 4.16. Also, since the user-specific model is pruned (and re-trained) only for user classes, it has significantly higher accuracy as compared to the user-agnostic model for the user-dataset. Furthermore, for the complete dataset with inputs belonging to outside user classes, the accuracy drops to $< 1\%$ on using the user-specific model, ensuring its correct behavior.

For symmetric pruning on Edge TPU, we show in Figure 4.17 that user-specific model size can be reduced by 2.6× from 21.7 MB to 8.3 MB, resulting in a speedup of 1.5×. The user-specific model also improves the accuracy to 73.6% for the user-dataset, within 1% margin, compared to the unpruned model accuracy of 72.4%, as shown in Figure 4.18. Similar to the mobile CPU platform, the accuracy drops to $< 1\%$ (close to zero) for the user-specific model on the complete Imagenet dataset. As discussed for the Inception-V3 model, there are two factors that contribute to performance improvement in Edge TPU. The first is the reduction of model size because of channel pruning, and the second is the reduction in the Image-to-column (Im2col) operation that is a part of the pre-processing step.

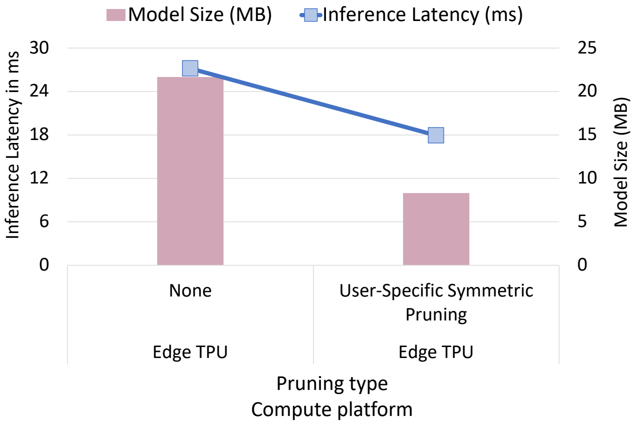


Figure 4.17: ResNet-50: Inference latency and model size for different pruning types on Edge TPU platform that supports Int8 precision for inference and BFP16 for pruning.

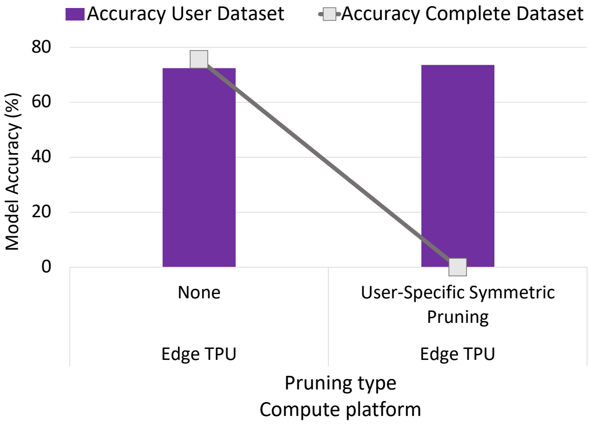


Figure 4.18: ResNet-50: Model accuracy for user-specific dataset and the complete Imagenet dataset for different pruning types on the TPU platform that supports Int8 precision for inference and BFP16 for pruning.

Pruning performance: In Table 4.3, we report the duration of the pruning phase, comprising 1,000 images. We find that the mobile CPU with asymmetric pruning can process 2.94 images/sec, which accumulates to a total time of 340s for the pruning phase. Edge TPU with symmetric pruning can process 10 images/sec, aggregating to 99.58s for the pruning phase. Our repurposed

Platform	Pruning Type	Precision	Pruning Phase Duration
mobile CPU	User-Specific Asymmetric	FP32	340.05 (s)
Edge TPU	User-Specific Symmetric	BFP 16	99.58 (s)

Table 4.3: Resnet-50 pruning comparison between mobile CPU and repurposed Edge TPU. Edge TPU is able to reduce pruning time by $\approx 3.42\times$. We expect the pruning to be a one-time cost for long inference phases where user classes remain stable.

Energy: We also observe improvement in the energy efficiency of computing models on mobile devices. The energy per inference reduces to 0.4J for the user-specific model, compared to 0.98J and 0.67J for the original and pruned user-agnostic model, respectively.

In the rest of the evaluation section, we only show results for the Inception-v3 model. The ResNet-50 model is made up of convolution layers similar to Inception-V3, and based on the above discussion, the behavior of user-specific models built from the two original models is coherent. Hence, trends and insights gained from the Inception-V3 model will be applicable to ResNet-50.

4.7.3 Adaptive System:

For the adaptive system, we start with a dataset that has only user classes as inputs. Upon building the user-specific model and utilizing it for predictions in the inference phase, we modify our dataset to include inputs belonging outside user classes. We gradually add outside-user-classes inputs in the 50 image window. For the first 10 images of 50 images, 1 out of every 5 inputs belongs outside of user classes. For the next 10 images, 2 out of every 5 inputs belong outside of user classes. Following this pattern, all the inputs lie outside user classes for the last 10 images of the 50 image window.

In Figure 4.19, we show the effectiveness of our proposed collaborative system on the above-discussed user trace for the Inception-V3 model. We show the model building process for the given user trace and also depict our system's behavior once user preferences start changing. At time t=0, the system kick-starts from the learning phase for a 50 input window. Once it learns user classes, it enters the pruning phase, where it uses the original model for inference while simultaneously

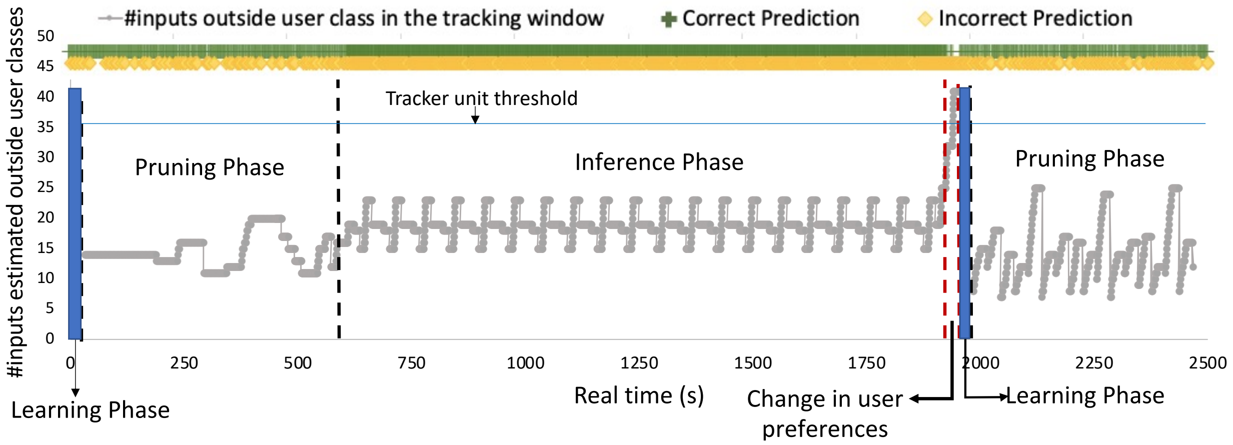


Figure 4.19: Trace showing MyML in real-time. We show the three phases - learning, pruning, and inference – of our end-to-end system as well as illustrate the working of the tracking unit that monitors the change in user preferences.

creating a user-specific model. Once the pruning phase is complete, we switch the original model with the newly created user-specific model for inference.

During all phases, our tracking mechanism checks for divergence in user preferences. For our experiments, the tracker counts the number of outside-user-classes inputs over a window of 50 input images and reports divergence if the count exceeds a threshold of 70%. In Figure 4.19, we show the running average over the tracking window. Since the tracker resets its count after each window, we observe a seesaw pattern in our trace. As shown in the figure, when the user slowly starts changing the preferences around 1900s in real-time, the tracker count shoots up and crosses the set threshold. The system then switches to a learning phase.

The trace also shows the correct and incorrect predictions by the appropriate inference model in each phase. We see that there is a drop in the correct predictions only in the period where user preferences transition to new classes.

4.7.4 Sensitivity Studies:

In this section, we conducted separate studies for an in-depth analysis of the MyML technique.

Layer breakdown:

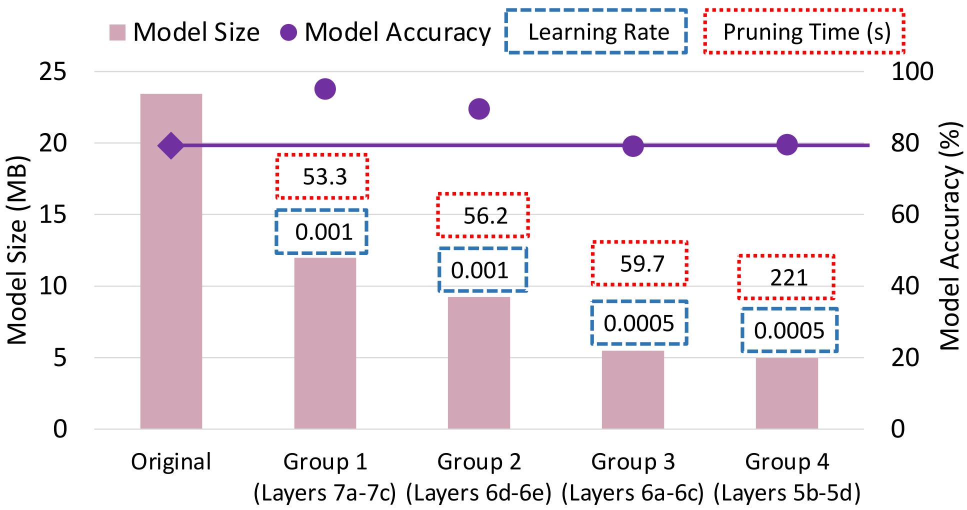


Figure 4.20: Per layer asymmetric channel pruning showing model size, learning rate, and pruning time for bottom-up pruning.

The aim of this study was to understand in detail the pruning rates, pruning times, and learning rates for individual layers comprising the deep neural network model.

Pruning rate: As shown in Figure 4.20 and Figure 4.21 for asymmetric and symmetric channel pruning, model size decreases as we prune more layers and move towards the top layers from group 1 to group 4. Asymmetric channel pruning has a higher model size reduction compared to symmetric channel pruning because asymmetric pruning is more flexible than symmetric pruning with constrained channel ID selection. However, both of these pruning types follow similar trends for each group of layers, as discussed below.

Group 1, consisting of layers 7a, 7b, and 7c, has the highest pruning rates, leading to a drastic reduction in model size. This reduction occurs because the bottom-most layers have a large number of filters and channels accumulating to 12 MB of model size, thus resulting in a significant model size reduction. Furthermore, bottom layers contribute towards the prediction stack, which makes them more amenable for pruning. The model accuracy increases on pruning group 1 because we tune the prediction layers for specific user classes, giving a significant boost to accuracy. As we move up in the model and prune the top layers, the model size and corresponding pruning rates slowly decrease until group 4. This behavior occurs because the top layers are smaller in size and built for feature extraction, making them less prune-able. Furthermore, as we move up in the

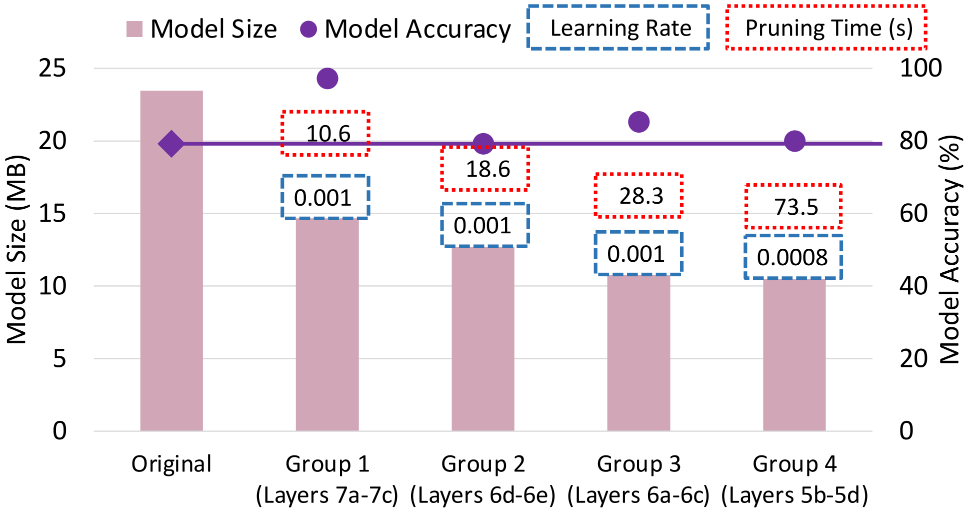


Figure 4.21: Per layer symmetric channel pruning showing model size, learning rate, and pruning time for bottom-up pruning.

model and prune the top layers, the accuracy drops with each group until it is similar to the original model accuracy. Moving from group 3 to group 4 gives a small reduction in model size; however, it stabilizes the error and makes the model more robust.

Pruning time: We also show the pruning time for each group of layers in Figure 4.20 and Figure 4.21. Pruning time increases as we move up in the model, following the bottom-up pruning technique, because while pruning layer n, all the layers from layer n to the last layer will be re-trained. For example, while pruning layers in group 4, all the layers in groups 1 to 3 will be re-trained. Thus, group 4, the top-most group of layers, takes a big chunk of time. This is because we train almost the entire model, except the untouched top feature extraction layer, and we train for one extra epoch to get a stable model. Furthermore, pruning time is shorter for symmetric pruning on the edge TPU accelerator as compared to asymmetric pruning on general-purpose mobile CPU.

Learning rate: Inspired by a commonly used training procedure that starts with a high learning rate for the first few epochs, which is lowered gradually for later epochs, we also form a learning rate schedule for the bottom-up pruning, as shown in Figure 4.20 and Figure 4.21. The bottom groups, comprising group 1 and group 2, have the highest learning rate of 0.001. Learning rates are reduced by 10-fold for the top layers in groups 3 and 4. Reducing the learning rate with

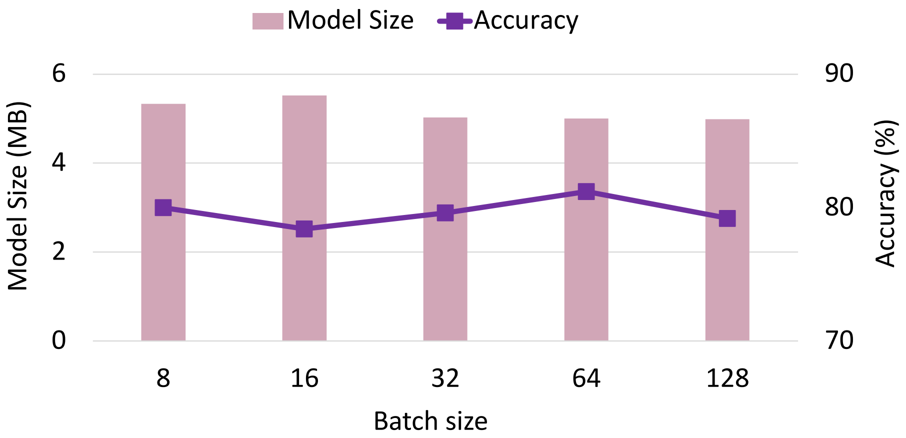


Figure 4.22: Model size and accuracy for increasing training batch size.

time/layers allows the model to vigorously learn and jump around various local minima during the start of the pruning process and gradually slow down to settle on global minima with a very low loss value.

We also observe a difference in learning rates between asymmetric and symmetric pruning for top layers/groups. The learning rates for top layers are relatively higher for symmetric pruning. We suspect this is because, for symmetric pruning, more error is accumulated due to floating point to block floating-point conversion as we move up to the top layers. Therefore, there is a need to have higher learning rates in order to evade local minima to make up for the extra error and stabilize to a low final loss value.

Training batch size:

Training batch size is an important parameter for the MyML approach to creating a user-specific model. The batch size determines the number of times we can update weights for a given number of images/inputs in the dataset. For example, a batch size of 10 for a dataset with 100 images will lead to 10 model updates, whereas a batch size of 25 for the 100 image dataset will give only 4 model updates. Though a smaller batch size can give more model updates, keeping the size too low can lead to the model jumping around different local minima and not stabilizing at a small loss value. Hence, there is a trade-off between batch size and the accuracy (robustness) of the model. In MyML, we want to keep a small batch size to have faster model updates within a reasonable amount of user data. Therefore, we present a sensitivity study with a batch size in the range of 128 to 8, as shown in Figure 4.22. In this study, we keep the number of updates constant (to 25); thus, the dataset size varies for training batch size. We observe that, at the largest batch size of 128 and dataset size of 3.2k, we can achieve the smallest model size (highest pruning rate) because the model has more data to learn and evade local minima to settle at a stable loss. A higher pruning rate demands more data to converge at a low loss value. As the batch size reduces, the model size increases (*i.e.,* pruning rate reduces) to stabilize at a low, stable loss with less amount of data. However, the difference between the model size is not very significant. A model created with a batch size of 8 is only bigger by 7%, compared to a model built with a batch size of 128. Across all the training batch sizes, we maintain an accuracy margin of 1% within a baseline accuracy of 79.2%. Therefore, for this work, we choose the batch size of 8 for training user-specific models that have an accuracy within 1% of the original unpruned model.

Scalability with number of user classes:

The second study measures the utility of building a small user-specific model over user-agnostic pruning as the user diversifies their preferences or choices. Therefore, we conduct a sensitivity study with an increasing number of user classes representing user preference. As done for prior discussed results, we train these user-specific models to maintain accuracy within a margin of 1%. Figure 4.23 shows that, even on increasing the number of user classes from 5 to 40, we achieve significant model reduction over user-agnostic pruning. For instance, with 40 user classes, MyML gives 1.5× and 2.8× reduction compared to the user-agnostic pruned and original model, respectively. This reduction in model size demonstrates the advantage of utilizing user-specific models over the original generic model even as the user expands their preferences. The increase in model size from 5 to 10 classes is 1.35 ×; this increase reduces to 1.13× from 10 to 20 classes and 1.1× from 20 to 40 classes. The increase in model size is highest when we expand from 5 to 10 classes, and thereafter, it tapers off as we expand to include more classes. Thus, we can infer that even as user preferences will expand beyond 40, the increase in user-specific model size will be by a small linear fraction compared to model size at 40 classes.

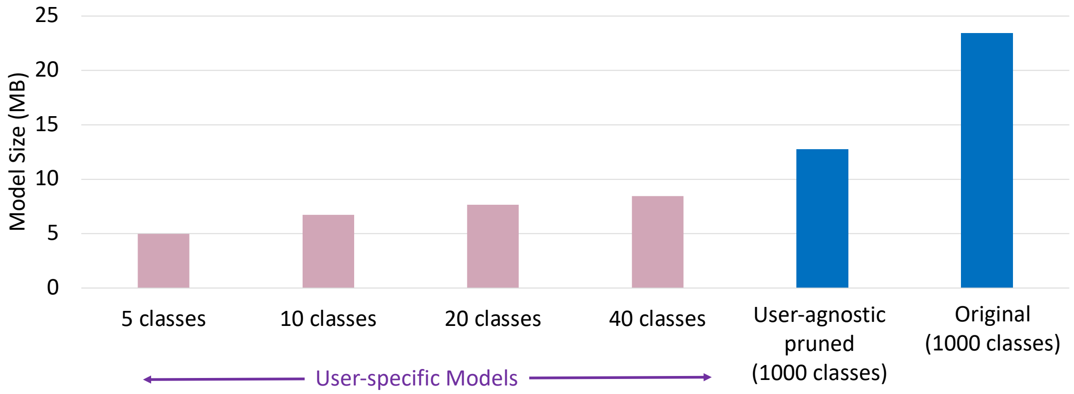


Figure 4.23: Scalability of user-specific models with an increasing number of user classes.

Ablation Study: We also performed an ablation study by choosing 5 known nearby classes – pickup truck, tow truck, trailer truck, tractor, and recreational R.V. – as user classes for building a user-specific model. We found that the model size for inception-v3 can be reduced to 6.2 MB from 23 MB while maintaining an accuracy of 79.2% for the user subset. This validates our hypothesis that, by leveraging user preferences, we can build tiny user-specific ML models to improve the efficiency of ML applications on user-devices.

4.7.5 Discussion

Re-training Cost: Although optimized by our MyML technique, pruning (re-training) is an expensive step in the process of building a user-specific model on a resource-constrained edge device. Thus, it is necessary to amortize the pruning cost over a long inference phase. As reported in the evaluation section, the average pruning duration phase for asymmetric pruning on the CPU is 365s or $\sim$3 mins. Thus, it is possible to prune (re-train) and create a new user-specific model every 3 mins. However, to amortize the pruning cost, we should run the inference phase for another 3 mins before we can start building a new user-specific model. Furthermore, for symmetric pruning on edge TPU, the average pruning phase duration is 116s or $\sim$2 mins. Thus, to amortize the pruning cost, the inference phase should be a minimum of 2 mins. Hence, if user preferences change as frequently as every 2-3 mins, MyML would not be able to create user-specific models at such a fast pace; however, this may not be the case for an average user with more stability in user preferences.

[122] shows a monthly change in user interests based on their twitter activities. Furthermore, [103] reports that the model retraining frequency can vary between hourly to multi-monthly for a wide range of ML models/tasks to capture a collective shift in preferences across all users, which will be slower for individual users.

Practical Application: To demonstrate the practicality of our approach, we apply a real-world dataset from Kaggle [18] to the Inception-v3 model to obtain the ratio of user classes and outliers. We test our method on 500 images. The learning phase operates on the first 100 images, and the remaining 400 images determine the fraction of user classes and outliers in the dataset. Unlike Imagenet, where each image is manually processed to have exactly one object/entity, the Kaggle real-world dataset has multiple objects in each image. Thus, we take the top-5 predicted classes for each image in our analysis, which account for 119 unique classes within the 500 image window. We mark the ten most frequent classes in the learning window as user classes and find that 95% of the remaining images belong to these user classes. Thus, we observe that the ten frequently appearing user classes (8.4% of total classes) consistently encapsulate 95% of images.

Our collaborative system has a tunable threshold for outlier tolerance. With a 95% threshold, the collaborative system can tolerate a maximum of 5% outliers before discarding the current user-specific model to create a new user-specific model for new preferences. We offer two solutions to handle outliers. The first solution sends the 5% outliers to the cloud server for computation, which applies the bulky original model, ensuring privacy for 95% of the inputs. The second solution is to infer the 5% outliers using the bulky original model on the local edge device. The second solution ensures privacy for all inputs by computing everything locally. Hence, MyML enforces privacy for 95% of user inputs regardless of the methods. Moreover, if the 5% outliers are computed locally on the edge device using the original model, we provide 100% privacy for all user inputs.

We show that for the collaborative system, which sends the 5% outliers to the cloud, the speedup is 2.2×. Additionally, computing the outliers on the device results in a 2.1× speedup. Note that the remaining 95% of the inputs are always computed at the edge device employing the user-specific model. These speedups are lower than the 2.3× speedup achieved when only the user-specific model is applied to all inputs without the differentiation of outliers or its extra computation.

4.8 Limitations

Though we provide an end-to-end holistic approach that learns, builds, and deploys user-specific models based on user preferences, there are still some limitations and scope for improvement for this work.

This work assumes that there is a pre-built accurate original model ready to serve the user that acts as ground truth. Training such a baseline model from scratch is a very expensive and time-consuming process. However, it is a one-time process that can be done in the back-end cloud server.

In this work, user-specific models are derived from original models, which are convolution layer-based deep neural networks. Hence, the user-specific models are, in turn, made of computationally complex convolution layers. There can be an alternate way to build smaller user models from scratch comprising simpler MLP layers and much lower depth. This approach has not been explored in this work. The above limitation can be further expanded to study the switching point from a simpler multi-layer perceptron (MLP) layer-based user-specific model to pruned convolution layer-based user-specific model. When user preferences belong to a few classes (*e.g.*, 5), simpler models may provide good accuracy with smaller model sizes. However, as user preferences expand to a large number of classes, simpler models might not be accurate, and we may need to switch to pruned complex models.

We determine user preferences by simply choosing the top-k appearing classes/categories in the learning phase window. Here, the value of k is static and pre-defined by the user or the vendor. This can be replaced by a sophisticated dynamic approach that is independent of k.

We use entropy and output probability to determine whether the inputs belong to user classes or outside user classes. Once the majority of inputs in a window are estimated to be outside user classes, we discard the current user-specific model and swap it with the original model. This approach hurts the accuracy of the input window based on which we detect divergence in user preferences. It can be replaced by a more fine-grained approach, where we can send individual inputs to the original model if they are marked as outside user classes. However, such a fine-grained system will require more robust statistics apart from prediction probability and entropy.

4.9 Conclusion

To circumvent the problems arising from offloading machine learning to the cloud, in this work, we presented MyML, a hardware-software solution that supports machine learning at edge devices. We leveraged the transfer learning approach to create small, lightweight, user-specific ML models based on user preferences instead of defaulting to a large, compute-intensive ML model. We proposed hardware-friendly, bottom-up pruning, which can be utilized by any mobile platform, and we also repurposed a systolic array-based edge accelerator to support user-specific transfer learning on edge devices without any cloud services intervention, thus ensuring user privacy. We also developed a collaborative edge system that tracks deviations in user preferences to switch back to the original model from the user-specific model and restart the model building process. We demonstrated that the user-specific model could reduce the model size by 4.7× and 2.25× using asymmetric and symmetric pruning for mobile CPU and edge TPU, respectively. In the next chapter, we aim to expand this approach of building user-specific models for recommendation systems, a new machine learning application that is the backbone of social-network platforms.

CHAPTER 5

User-Driven Recommendation Systems

Until now, we have studied and optimized many tasks widely used by popular edge applications, such as GPS tracking, video recording, and image recognition, with the motive of improving the application's efficiency. In the last chapter, we specifically explored the potential of creating smaller user-specific models based on user preferences for image recognition tasks utilized by smart-cameras apps, galleries, and image-based social media apps like Pinterest, Instagram, Facebook, etc.

In this work, we expand our application suite to include recommendation systems that constitute a significant portion of the inference cycle [221]. Recommendation systems are an integral part of popular mobile applications, including social media (e.g., Instagram, Facebook, Pinterest), online shopping (e.g., Amazon, eBay, Alibaba), movie recommendation (e.g., Netflix, Hulu, Prime), fitness (e.g., Fitbit, Peleton), and e-learning (e.g., Coursera, Udemy). Netflix attributes 80% of its streaming hours [10] to the recommendation system, and 35% of Amazon purchases [16] result from the recommendation algorithm. Recommendation systems are the top consumer of AI compute cycles in production-scale datacenters [97].

Furthermore, the steep rise of mobile devices and digital content in the last decade has increased the demand for higher quality and more efficient recommendation systems to enhance user experience. Hence, there are numerous ongoing efforts across the scientific community to improve the efficiency of recommendation systems [133, 52, 220, 95, 184, 146, 81]. For example, prior works [133, 52, 184, 146] have focused on using hot-embedding caching or near-memory solutions and utilizing multi-stage recommendation models [96] to improve overall efficiency.

Until now, recommendation systems have been considered to be datacenter only applications because of their large memory footprint. The existing solutions miss the opportunity to utilize user history information, which is readily available on edge devices. Consider a recommendation *query* that consists of several inferences (100s) to rank candidates. Each inference has to compute MLPs and lookup multiple embedding tables. We observe that a significant portion of an inference specific to user data is *repeated* for all inferences in a query, as shown in Figure 5.1. This common computation can be performed once per query on edge to boost overall efficiency significantly.

Deriving from this insight, in this work, we present *Duet*, a collaborative user-driven edge-cloud recommendation system. Duet decouples the monolithic recommendation model into two smaller concurrent models, user and item models, distributed on edge and cloud, respectively. The user model operates on the user data at the local edge device and transfers its output to the datacenter. Simultaneously, the item model computes item features for all candidates to be ranked. It then combines item model outputs with the user model output to provide final recommendations. More importantly, unlike the item model that computes item features for all the candidates to be ranked, the user model needs to be computed only once for the recommendation request. Thus, we reuse the user-specific computation across all the candidates being ranked. Further, by processing user-specific information on edge, *Duet* enhances data privacy.

The two models, the user and item models, are created by splitting the *three components* of a generic recommendation model – embedding tables, the bottom MLP stack, and the top MLP stack. *First*, embedding tables are divided between edge and cloud. We observe that storing the entire user embedding table on the edge is overkill. Compared to large user embeddings (GBs) on the cloud that cater to millions of users with diverse histories, user-specific embeddings at the edge device store entries pertaining to only one user's history, reducing storage needs drastically (KBs). *Second*, we split the bottom MLP stack between static and dynamic inputs. Static inputs comprise stable user information (such as age, gender, device id, device make, device IP address, number of apps, etc.), and dynamic inputs are constituted by the transient context inputs (such as

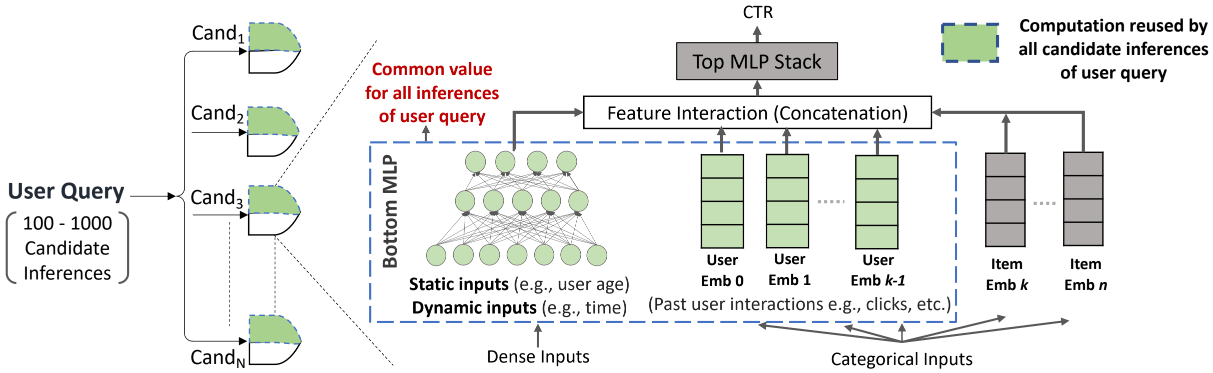

Cand1
Cand2
Cand3
CandN
User Query
100 - 1000 Candidate Inferences
Common value for all inferences of user query
CTR
Top MLP Stack
Feature Interaction (Concatenation)
Computation reused by all candidate inferences of user query
Bottom MLP
Static inputs (e.g., user age)
Dynamic inputs (e.g., time)
User Emb 0
User Emb 1
User Emb k-1
(Past user interactions e.g., clicks, etc.)
Item Emb k
Item Emb n
Dense Inputs
Categorical Inputs

Figure 5.1: Recommendation model comprising of three components – bottom MLP, embeddings, and top MLP – computes 100s of inferences to rank all the candidates of a query. Common computation pertaining to user-specific information can be reused across all the inferences of the query.

current time, day, and time since the last watch). The static bottom MLP is pre-computed, and its output is reused during execution; only lightweight dynamic bottom MLP is computed in real-time. *Finally*, the top MLP is also split into user-specific and item-specific layers. User-specific layers are computed once per query on edge. Its outputs are aggregated with outputs of item-specific layers, which are channeled into final layers to produce the Click Through Rate (CTR) probability.

Since the decomposed user model is computed on a small edge device, we present a lightweight Duet architecture to improve the edge user model's energy efficiency. We propose a hardware unit with a small user-specific scratchpad and location tracker to capture up-to-date user states in hardware. Further, we support memoization for pre-computed static bottom MLP results and heterogeneous edge-friendly lower precision (qint8) format for user-specific top MLP layers.

We evaluate our proposed end-to-end collaborative system across five recommendation models and boost the performance by 6.4× and the energy efficiency by 4.6× with the accuracy impact limited to $<= 0.06\%$.

5.1 Background and Motivation

The recommendation is a two-stage process – candidate generation and candidate ranking. Candidate generation is tasked to provide potential candidates that the user may like; to do so, it uses

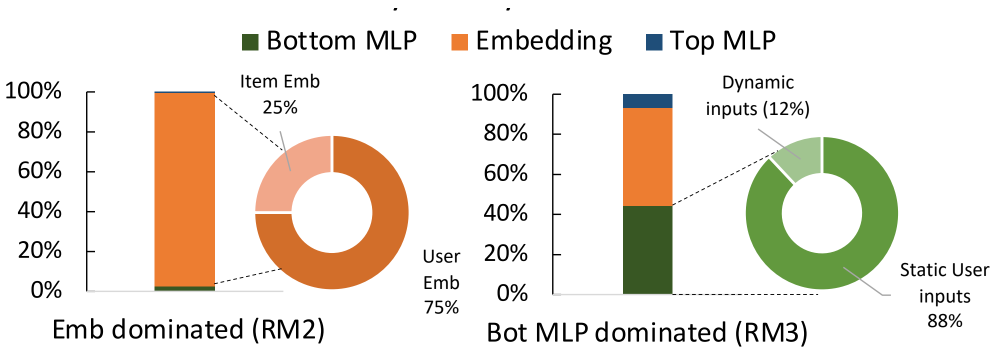


Figure 5.2: Timing Breakdown in three components for a query with 1024 candidate inferences for two RM Models.

lightweight filtering kernels like KNN or Matrix factorization. The candidates generated from this step are then processed by the candidate ranking step, which utilizes a heavy DNN-based recommendation model to rank all the candidates and recommend the top candidates with the highest CTRs to the user. Thus, each recommendation query requires computing inferences for hundreds to thousands of candidates (termed as query size) on a heavy DNN-based ranking model, which makes this task very compute-intensive. Moreover, tight latency targets (SLA) to ensure a smooth user experience makes it latency sensitive.

Recommendation models are built using embedding tables and MLP layers, as shown in Figure 5.1. The models have three components – the bottom MLP stack to process continuous inputs, embedding tables to process sparse categorical inputs, and the top MLP stack – to predict the click-through rate (CTR). Prior works [97, 95] have demonstrated that the models can either be embedding-dominated or MLP-dominated. We further break down the timing and illustrate in Figure 5.2 the time consumed by the three components for two particular model definitions. We realize that while model RM2 is embedding-dominated, RM3 is dominated by the bottom MLP stack. Thus, each of the components has to be optimized to ensure performance improvement for all the models.

We observe that previous works [118, 146, 133, 96, 95, 165, 220, 52, 184, 128, 150] consider all the embedding tables to be equivalent, which may not always be the case. There are user embedding tables that gather past user history and item embedding tables that extract item features for the candidates. As shown in Figure 5.2, the bigger chunk (75%) of the embedding time is consumed by user embeddings, while item embeddings consume the remaining 25% of the embedding time. Our findings yield two discoveries. First, we discover an opportunity to reduce embedding table memory accesses because user embedding accesses, corresponding to a user history, are common for all the inferences performed to rank multiple candidates of a user query. Hence, the user embedding can be processed once for a query and shared by all the inferences. Second, we discover that we can decouple the two embeddings, and user embedding operations can be offloaded to user edge devices that have all the information about past user activities.

We also observe that, similar to embeddings, all the inputs to the bottom MLP stack are not equivalent. A significant fraction of the inputs corresponding to stable user profile information can be computed once per query and do not frequently change for a user. As shown in Figure 5.2, as much as 88% of the bottom MLP inputs correspond to static user information, which is readily available on the user edge device, and, thus, can potentially be computed locally.

In this work, Duet, we leverage this non-uniformity in the model definition to split the model into two concurrent models: one to process user information, which is computed once per query and shared by all the candidates of the query, and the other to process individual candidate information for all the candidates to be ranked. We propose a holistic approach that tackles all three components of the recommendation models by: 1) reducing and distributing the model on the edge and datacenter and 2) optimizing the user model for the resource-constrained edge platform. To the best of our knowledge, this is one of the first efforts that leverage user preferences to incorporate edge computation in the recommendation engine.

Prior work EdgeREC [85] shows the advantage of using extra user inputs available only on the user device, like scroll speed and exposure duration, to improve user-engagement/accuracy. Our solution incorporates user edge devices to improve the performance and energy of the recommendation engine. A related prior work, RecPipe [96], proposes a multi-stage model by adding a filtering model before the recommendation model at the datacenter. In this work, we decompose the recommendation model itself into smaller concurrent models, one of which leverages user information by processing on the edge device, thus offering more privacy.

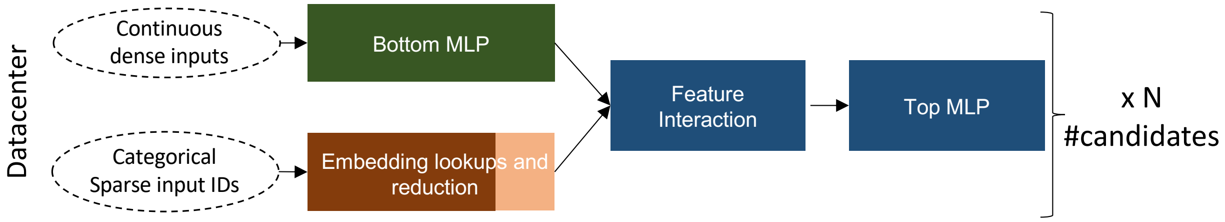


Figure 5.3: Data flow for the state-of-the-art recommendation system at the datacenter computing N inferences for the N candidates.

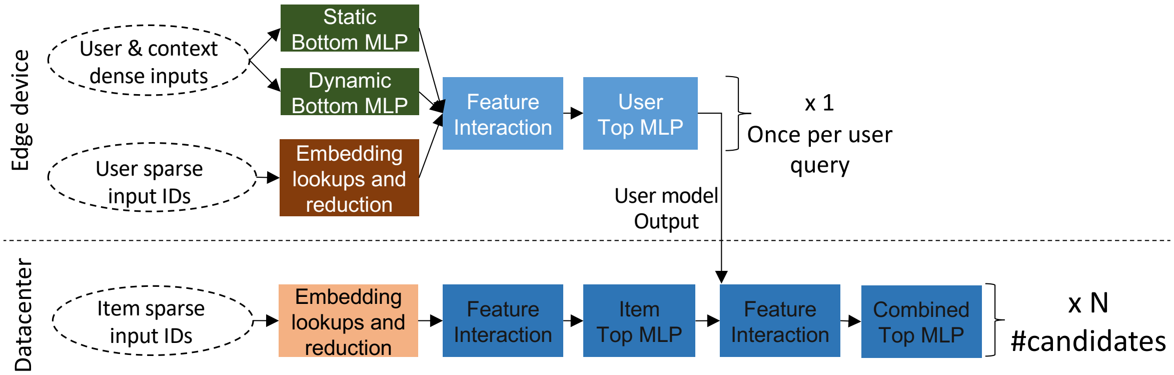


Figure 5.4: Data flow for our proposed collaborative edge-cloud recommendation system. The monolithic model is decoupled into two concurrent models running on the edge device and datacenter. The edge model is computed once per user query, and the reduced datacenter model is computed for N candidate inferences.

5.2 Proposed Collaborative Recommendation System

The data flow through the three components of the recommendation model is illustrated in Figure 5.3. The bottom MLP layers process the continuous dense inputs to produce a k-dimensional output vector, and the embedding tables process the categorical inputs sparse IDs. Embedding tables map sparse input IDs to a dense k-dimensional vector. Each embedding table gets multiple sparse input IDs, over which it performs gather and reduction to produce a k-dimensional output vector. The feature interaction block concatenates the output vectors from embeddings and bottom MLP to give a final feature vector, which is then processed by the top MLP layers to output the Click-through rate (CTR). This recommendation model inference is performed repeatedly N times for each N input candidate to predict CTR for all the candidates.

In this work, we propose a user-aware recommendation system, which decomposes the giant monolithic model into a user model at the edge device and an item model at the datacenter based on the input characteristics, as shown in Figure 5.4. The user model at the edge device computes user inputs comprising: 1) user-history-related sparse inputs IDs, 2) dense user inputs, and 3) dense context inputs. The information processed by the user model is common to all the inferences that need to be computed for ranking the N candidates of the query. Thus, this common computation is processed once for a query and is reused for all the N candidate inferences, thus, significantly reducing the embedding table memory access and MLP computation. The output produced by the user model is sent back to the datacenter for further processing.

Simultaneously, the item model at the datacenter individually processes each candidate's sparse input IDs to extract candidate features. Thus, N inferences are performed on the item model for N candidates. Though N inferences are still computed on the datacenter, the datacenter performs less work by offloading user inputs processing to the edge platform. Finally, the output feature vector of the user model is aggregated with each of the N candidate feature vectors individually. The final MLP layers then process the aggregated N feature vectors to produce CTR for all the N candidates of the user query. In a datacenter environment, multiple small queries can be merged for batch processing, or large queries can be sharded into smaller sub-queries to achieve optimal throughput and latency. Therefore, the baseline model batches any random combination of candidates; however, for our proposed solution, we batch only the candidates belonging to the same user query with the vision of extending it to a random combination, using unique tags for queries. We leave this extension for future work. Based on a production query size distribution [95], the 75th percentile query size is between 200-250, which is able to saturate the system resources with queries following poisson distribution arrival rates. Based on our benchmarking, the RM2 model with a query size of 256 can barely serve 49 QPS within the SLA targets. Thus, a median query size of 256 is large enough to keep system utilization high.

5.2.1 Model Decomposition:

In this section, we discuss in detail the decomposition of the three components of the recommendation model across edge and cloud.

5.2.2 Embedding tables

The size and the number of embedding tables in recommendation models are constantly increasing. User embeddings comprise a major fraction of the total embeddings. For example, $n-1$ lookups out of the n embedding tables lookups pertain to past user-interaction [48, 128]. Recent work [81] estimates user embeddings to occupy as much as $\approx 75\%$ of total embedding memory space. User embeddings are accessed to retrieve user information, such as past user likes, favorites, clicks, etc. A user-agnostic model, which does not differentiate between user and item embedding characteristics, accesses the user embedding tables every time a recommendation inference is calculated. However, for a given query that ranks multiple candidates, past user interactions are common across all the candidate inferences; thus, we can avoid repeated user embedding lookups over the entire query by computing them once and reusing them across all the inferences. For Duet, we first leverage this user embedding property to decouple the user embeddings from item embeddings, where user embedding is computed only once for a query.

Second, we also discover the opportunity to maintain small user embedding tables on the local edge device instead of accessing the giant user embedding table at the datacenter. The user embedding tables at the datacenter (GBs) store cumulative histories for millions of users with diverse preferences. However, user embedding memory accesses corresponding to *one* user history (KBs) is required for a given user query. Assume a scenario with three users, user1, user2, and user3, with individual user histories. User1 has liked movies 1-10, user2 has liked movies 5-30, and user3 has liked movies 20-50. In the datacenter, the movie embedding table stores entries for all the movies from 1-50. When a query from user1 is being served, the model will access this entire table to lookup only at user1's past liked movies from the entire table. Furthermore, unlike this example where user1 accesses contiguous rows 1-10 of movie embedding tables, embedding table access can be random and spread all over the embedding table. These factors make embedding operations very expensive.

Nevertheless, based on this insight, we propose storing user-specific embedding tables pertaining to user history on the local edge device. This shrinks the embedding table size from GBs to KBs, which is feasible on a small edge device. Further, as shown in Figure 5.5, we not only reduce the user embedding table size but also store embeddings in a contiguous format instead of accessing them randomly across the entire embedding table. Overall, we decrease the number of embedding lookups from history_size×num_emb_tables (n×k) to history_size×1 (n×1). The user history is bound to change with time as the user interacts with more items. In Duet, we constantly update the user-specific embedding tables with recent user interactions with the help of Duet's location tracker in the hardware unit, as described in detail in architecture Section 5.2.6. On a high level, for our edge system, the embedding entry data is relayed along with the meta-data attached to the items recommended to the user. For example, as shown in Figure 5.5, a movie recommended to a user will come along with the particular movie embedding entry and its genre embedding entry. Each embedding entry is only a few bytes (128 bytes). If the user clicks the movie, we update the user-specific embedding table with the movie embedding entries associated with it. The location tracker maintains a user history of fixed length n, corresponding to past n lookups in the local device memory. Consider n is 10, and the user-specific embedding table contains the past 10 liked

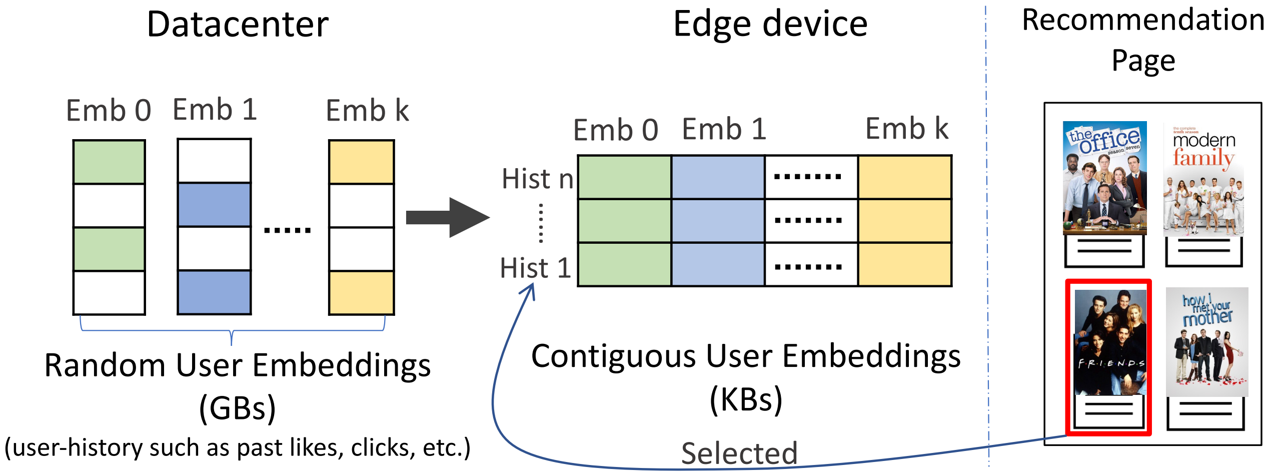


Figure 5.5: Converting randomly accessed large user embeddings at datacenter to contiguous and small user-specific embeddings at edge device, which are continuously populated and updated with recent user interactions.

movie entries. When a new movie 11 is liked by the user, the oldest entry in the table (movie 1) is removed, and it is populated with movie 11 embeddings. In this manner, we constantly update the user history stored in the user-specific embedding tables in a circular fashion on the edge device. We discuss in detail the working of the location tracker in a later section describing the specialized hardware unit.

5.2.3 Bottom MLP stack

Bottom MLP usually has a depth of three to four layers, and it processes continuous dense features to provide an output feature vector, which is concatenated with the outputs of embeddings. A significant fraction of the dense inputs to the bottom MLP layers is composed of stable user information, which does not frequently change, such as user age, gender, demographic, number of user engagement sessions, number of apps on the user device, user device class, device os, device IP, etc. The remaining dense inputs comprise dynamic context inputs, such as time of day, day of the week, or time since the last watch/login. Again, the dense inputs are common for all candidate inferences of query and do not change while a query is being served. Thus, the common computation can be performed once per query and reused by all recommendation inferences of the query.

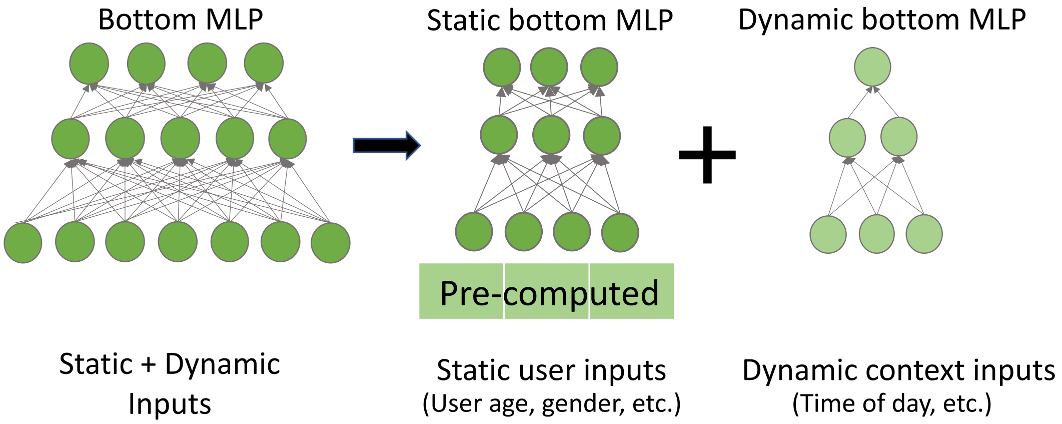


Figure 5.6: Decomposition of the bottom MLP stack into a pre-computed static bottom MLP and a smaller dynamic bottom MLP stack.

This optimization effectively reduces the amount of MLP computation required to process all recommendation inferences of a query. Since dense inputs comprise user and context information, which are readily available on the user edge device, we incorporate the bottom MLP in the user model computed on the edge device, as presented in Figure 5.4 showing our proposed data flow pipeline.

Static-Dynamic bottom MLP split: Edge devices are not friendly to compute-intensive kernels like bottom MLP. Computing bottom MLP in its entirety can be expensive on the small edge device and can hurt the efficiency of bottom MLP-dominated models. However, as mentioned above, the user information that is fed into the bottom MLP is relatively stable and does not frequently change, whereas dynamic time-dependent context inputs are different for every query. Based on this insight, in Duet, we split the bottom MLP into two MLPs – static bottom MLP and dynamic bottom MLP – as shown in Figure 5.6. Furthermore, the stable or static user inputs processed by the static bottom MLP layers can be pre-computed, and the output vector can be stored in a memoization register for future reuse, as explained later in the Duet hardware architecture Section 5.2.6. During query execution time, we directly use the pre-computed output vector of the static bottom MLP stack, which significantly reduces the amount of computation during runtime.

In contrast, contextual inputs of the bottom MLP need to be computed by the dynamic bottom MLP for every query. Dynamic inputs represent the state of the input parameters when the query is being served. They are a smaller fraction of total dense inputs and, thus, require a proportionally

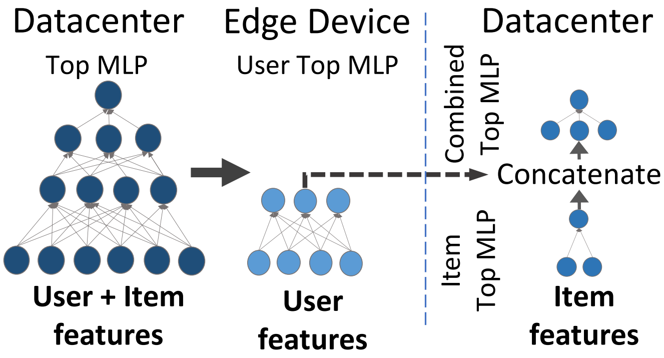


Figure 5.7: Top MLP stack decomposed into user top MLP, item top MLP, and combined Top MLP.

smaller MLP, reducing the amount of work performed in real-time at the edge device. We construct the proposed split bottom MLPs by dividing all the layers in proportion to the number of static and dynamic inputs. However, simply splitting the original bottom MLP based on input definitions and using it as is degrades the accuracy of the recommendation model. This is because the all-to-all connections of the MLP layers are broken during model decomposition. Hence, we re-train the decomposed model to regain the accuracy, as discussed in Section 5.2.5. The decomposed model has two separate bottom MLP stacks – static bottom MLP and dynamic bottom MLP. Post-training, the two stacks learn new weights that are independent of each other, and their outputs are concatenated before being processed by the next model component. We also observe that since the bottom MLP stack is responsible for feature extraction, it is not very sensitive to decomposition and is able to easily learn the new model structure. After re-training, the accuracy drop because of model splitting is limited to <0.03%, as discussed in evaluation section 5.4.3.

5.2.4 Top MLP Stack

The top MLP stack used for CTR prediction processes the concatenated output feature vectors of the bottom MLP and embeddings. It has a depth of three MLP layers followed by a softmax layer, which processes aggregated vectors to produce CTR for all the candidates. The top-K candidates with the highest prediction probability are then displayed to individual users.

User-Item top MLP decomposition: In Duet, we also strive to split top MLP as much as possible to distribute the work between the edge and the datacenter, as shown in Figure 5.4. However, since top MLP should consider the interaction between user and item features to predict accurate CTR, we cannot entirely split top MLP into two as done previously for bottom MLP. As demonstrated in Figure 5.7, we only split the first one-two layer of top MLP between the user model at the edge and the item model at the cloud. The remaining layers of top MLP layers remain in the cloud.

By splitting the first few top MLP layers, the user top MLP layers, which process user features on edge, are computed once per query. This would otherwise be computed for all the recommendation inferences of the query. Furthermore, by splitting the original top MLP layers, we can utilize reduced/smaller item top MLP layers to repeatedly compute all the recommendation inferences at the datacenter. Thus, splitting the top MLP layers helps distribute the work and reduces the overall work that needs to be completed to rank all the query candidates.

As shown in Figure 5.4, the output from the user top MLP at the edge is sent back to the datacenter. At the datacenter, it aggregates with the output vector of the N candidates computed by the item top MLP. The combined top MLP layers then compute the final aggregated vectors. The first few top MLP layers are split proportionately to the input vector length. The user top MLP, which processes concatenated outputs of bottom MLP and user embeddings, receives a major fraction of the split top MLP. In contrast, the item top MLP, which processes only outputs of item embeddings, receives a smaller fraction. However, the user top MLP is computed just once compared to the lighter/smaller item top MLP computed for all recommendation inferences. Thus, we provide a balanced work distribution by proportionately dividing the model. To avoid any significant accuracy impact of the split, we train the decomposed model as discussed in Section 5.2.5.

Processing top MLP on user features at the edge provides additional advantages by reducing the amount of data transferred from the edge to the cloud, resulting in energy and performance benefits. The payload size of the data uploaded to the cloud is important because the upload speeds for even the latest ultra-wide 5G is limited to 24 Mbps, which is 10x smaller than the 5G download speed of >300 Mbps. The evaluation section will discuss the communication overheads and their impact on energy and performance.

5.2.5 Training Decomposed Model

Decomposing the monolithic model by naively splitting layers degrades accuracy. To regain accuracy, we train the decomposed model from scratch offline in the datacenter. The decomposed models do not fine-tune MLP weights or embeddings for individual users. For example, if both users 1 & 2 like movie-1, they will have the same embedding value of movie-1 in their personal user histories on edge. Similarly, the weights of decomposed top and bottom MLP stacks are generic. For example, the static and dynamic bottom MLPs are generic to all users. However, decomposition allows the pre-computation of stable user-specific inputs by the static bottom MLP. Owing to its generic nature, we train decomposed models once at the datacenter. Trained decomposed models serve thousands of incoming queries, amortizing the training cost.

We require the definition of model inputs to determine the split ratio [user_inputs:item_inputs], which is utilized to split model components. For example, a split ratio of 0.6 on an MLP layer with 100 nodes will split user MLP and item MLP layers to have 60 and 40 nodes each. The split ratio is determined based on the inputs pre-defined in the original model, which are independent of users. Hence, the split ratio is not user-specific. The decomposed model is generic and can be offloaded on any user device. While the split ratio guides the decomposition of layers, the decision to split or not is empirically determined based on accuracy impact. In Duet, we iteratively split and re-train one layer at a time, starting from the bottom-most MLP layer up to the top layers, until we are within the pre-set accuracy drop threshold ($<0.1\%$). If the accuracy drop exceeds the threshold, we stop the splitting, thereby minimizing the accuracy impact. We observe that bottom MLP layers responsible for feature extraction are more friendly to splitting. However, top MLP layers responsible for prediction are vulnerable to splitting.

5.2.6 Duet Architecture

By decomposing the model, we add the user model and its associated work to the edge device, which would otherwise offload the entire query computation to the cloud. Computing the user model on the device without hardware optimizations leads to significant energy implications. We observe that the battery life reduces drastically from 24 hrs to 18 hrs (a 25% drop), if we compute the user model for a recommendation query every min. Hence, to reap the benefits of model decomposition, we require a more energy-efficient and practical user model for the resource-constrained edge device. In this work, we propose a specialized hardware unit, which is placed near the CPU and communicates with it directly, as shown in Figure 5.8. For a query, the CPU triggers a read request to the memoization register and the scratchpad of the hardware unit. The read values are then processed by the CPU. The CPU computes dynamic bottom MLP and embedding reduction. Their outputs are then concatenated and processed by the top MLP. We utilize heterogeneous lower (qint8) precision for top MLP to reduce its processing overheads and payload size while curtailing the impact of lower precision on accuracy. The output of the top MLP is transferred to the cloud for final processing. Additionally, if the values in the hardware unit need an update, the CPU sends a write request to the memoization register or (and) the scratchpad of the hardware unit. The write update request is processed in the background and does not interfere with the query serving process. We now explain individual components in detail.

5.2.6.1 Hardware Unit

The specialized hardware unit is responsible for storing all the relevant local user information required by the recommendation query. It comprises memoization support to store the output of pre-computed static bottom MLP and a user-specific scratchpad with a location tracker.

The memoization register stores the partially computed result of bottom MLP, corresponding to static bottom MLP, which is reused across queries. During execution, the memoization register value is read and concatenated with other model components. If inputs to the static bottom MLP change, the register value is updated in the background without interfering with the query execution

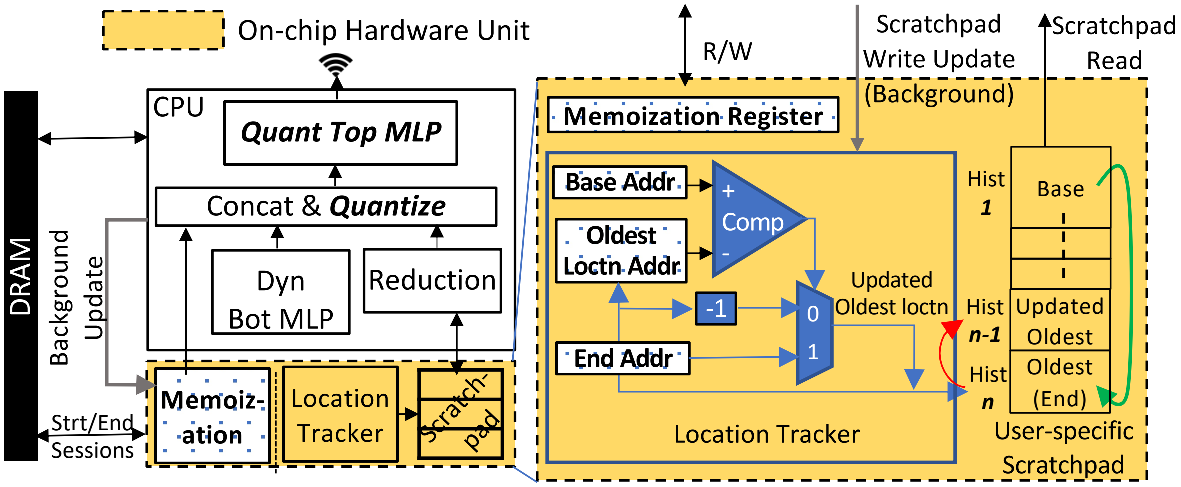


Figure 5.8: Proposed Duet architecture with an on-chip hardware unit and quantized int8 precision support to efficiently process the user model at the edge.

flow.

User-specific Scratchpad: Decomposing embedding tables into user and item embedding allows us to store only user-specific embedding tables on the device. However, we still have to access off-chip memory to fetch user embeddings into the CPU. Since the user-specific embedding table size is in the order of KBs that stores only a *fixed length* of history, we propose a user-specific on-chip scratchpad residing in the hardware unit to eliminate the DRAM accesses altogether. The scratchpad should contain only the most up-to-date history; thus, it requires a mechanism to populate and update the scratchpad as described below.

Scratchpad Population: User embeddings are related to past user activities, such as apps installed on the device, past liked movie genres, past clicked movies, etc. At the start of a session, the scratchpad is pre-loaded with the user embedding table, base address [*Base Addr*] & end address [*End Addr*] registers, and oldest embedding location address register [*Oldest Loctn Addr*] in the scratchpad from the previous session. As the session unfolds, we update the scratchpad with user embeddings corresponding to recent user interactions of the current session. If the user clicks on a new item, we update the scratchpad with the item's associated embeddings. There will never be a scratchpad miss because the history size is fixed to the past n lookups. Consider the previously discussed example of movie recommendation where n is 10, and the scratchpad contains the past

10 liked movie entries. When a new movie 11 is liked by the user, the oldest entry in the scratchpad (movie 1) is removed, and it is populated with movie 11 embeddings. Simultaneously, movie 2 becomes the oldest entry in the scratchpad.

Scratchpad Update: We maintain an up-to-date user history on the small scratchpad with the help of a lightweight location tracker, as shown in Figure 5.8. It provides the address of the oldest embedding location on the scratchpad, which is overwritten on a scratchpad update to keep a fixed history of length n. Tracker simultaneously also updates the oldest location address to the next oldest location, which will be written next. To update *Oldest Loctn Addr*, the hardware location tracker compares the values of *Base Addr* register and the current *Oldest Loctn Addr* register. In case of a mismatch given by a comparator, *Oldest Loctn Addr* is decremented or moved up (red arrow), pointing to the next oldest location. However, in case of a match, we have to circle back to position n (green arrow) and update the *Oldest Loctn Addr* register with the *End Addr* of the scratchpad. At the end of the current session, the scratchpad entries and the tracker state is stored in the main memory, which a later session can retrieve. In this way, we continuously populate and update the scratchpad to maintain recent user history in a lightweight manner. The update action is performed in the background whenever a user clicks on a recommended item at the end of query execution and does not interfere with the execution flow. The scratchpad will have the most updated history during the execution, and we simply look up (read) all the entries to retrieve user embeddings.

Duet architecture can easily support multiple models or apps sharing the scratchpad. Different models can store their hardware unit state in the memory, which is retrieved when a session starts to warm up the hardware unit. Scratchpad is also available to other applications in the absence of any recommendation sessions on the device. Thus, we enable high utilization of the expensive silicon area occupied by the hardware unit.

5.2.6.2 Heterogeneous precision with int8

Lower precisions are the most prominent solution for faster and more energy-efficient ML models. Splitting MLP stacks on multiple platforms presents an opportunity to have different precision on each platform, especially for resource-constrained edge devices that prefer int8 precision for performance and energy efficiency. Since the major fraction of the bottom MLP stack is precomputed and does not need real-time computation, it does not consume significant energy or time. However, user top MLP is computed for every query in its entirety and also constitutes the major portion of the top MLP stack, thus consuming more energy and time. We observe that by reducing the user model's top MLP precision from fp32 to quantized int8 precision, we can reduce the MLP energy (time) and the data-transfer energy (time) because of payload size reduction. Thus, we utilize the post-training static quantization int8 (qint8) precision for the user top MLP to convert it to int8 precision for inference. The concatenated output of the bottom MLP and embeddings is quantized during execution and processed by the quantized user top MLP to produce an output in qint8 precision, which is communicated to the cloud. Hence, quantization reduces computing and communication energy and time.

5.2.7 Multiple device synchronization

A user may have the same recommendation app on multiple devices, such as a smartphone and laptop, which leads to each device viewing only partial user history. Thus, there is a need for synchronization between devices. For all practical purposes, recommendation sessions for a user owning multiple devices will be in a serial fashion. Once a session is closed on device A, it will transfer its history via Bluetooth or wi-fi network to the next session on device B. An alternative is to use the cloud as the point of synchronization, where at the end of the session, device A will send its partial history back to the cloud, which will be communicated to the next device B. The communication overheads are limited to the size of partial history, which will be a fraction of the KBs of the complete history. In our diverse set of models, 4 out of 5 have a user history of <60KB, and only the fraction corresponding to a partial view of a few KBs is needed for synchronization.

Model	Bottom MLP	Top MLP	#EMB tables	#Lookups per Emb	EMB size (#entries)
RM1 [97, 95]	128-64-32	256-64-1	8	80	4M
RM2 [97, 95]	256-128-64	128-64-1	32	120	500K
RM3 [97, 95]	2560-1024-256-32	512-256-1	10	20	2M
RM4 [synthetic]	512-256-32	2560-1024-1	10	20	2M
RM5 [237]	-	200-80-2	3	1-200-200	1M

Table 5.1: Model configurations used for evaluation.

	Samsung S10e (Snapdragon 855)	X86 Server
Cores	1xA76 @2.84 GHz, 3xA76 @2.41 GHz, 4xA55 @1.78 GHz	18 @3.7Ghz
L1 cache	1x128KB, 3x128KB, 4x128KB	32KB
L2 cache	1x512KB, 3x256KB, 4x128KB	1MB
L3 cache	2MB	25MB
SIMD	NEON	AVX-512
DRAM	6GB, 34.1GB/s	90 GB, 80 GB/s

Table 5.2: Architectural specifications for mobile and server.

Importantly, this synchronization can be done at the end of a session and not necessarily during execution. The overheads are related to the transmission energy, which for a 10KB over Bluetooth network of 1W/3Mbps is 0.027J.

5.3 Methodology

We evaluate our solution for the five models listed in Table 5.1. Each model has a different bottleneck. Models RM1 and RM2 are embedding-dominated, RM3 is bottom MLP-dominated, and RM4 is top MLP-dominated. RM5 is also embedding-dominated but does not have dedicated item embedding tables. The last two embedding tables double as user and item embeddings, i.e., out of 200 embedding table lookups, 199 correspond to the user history, and one corresponds to the candidate information. The first embedding table is for user_ids; thus, an incoming user query with a unique user_id will perform one lookup on the user_id embedding table.

The baseline is evaluated on a server-class machine, and mobile performance results are evaluated on a Samsung 10E platform. The configurations are listed in Table 5.2. To support our techniques, we extend an open-sourced caffe2-based benchmarking framework provided by prior work [95]. We enable multi-threading for the caffe2 models to utilize the multi-core capabilities of the servers. To support the mobile platform, we extend the framework to export x86 caffe2 models to ARMv8 caffe2 models. ARM Caffe2 models use QNNPACK as the backend library to support quantized int8 precision on the mobile platform. To emulate the scratchpad's behavior on the edge device, we store the user-specific embeddings on the L3 (System) cache of Samsung S10E, our edge evaluation platform. The location tracker is synthesized separately using a 15nm commercial library. We report its post-synthesis timing, area, and power in Section 5.4.6. We utilize the speed benchmark utility provided by caffe2 to determine execution time and timing breakdown. Our programming model batches all inferences of a query and uses multi-threading to distribute the work on all cores of the server. Simultaneously, the mobile platform performs its user model computation once for a query.

To study accuracy impact, we train models RM1-3 for the Criteo Kaggle advertising dataset [8] using an open-sourced DLRM framework [174] and model RM5 for the Amazon electronics dataset [167] using an open-sourced DIEN framework [11]. The RM1-3 models are trained for MLP stack definitions listed in RM1-3 in Table 5.1. Based on Alibaba's Taobao dataset [1, 30] that explicitly state input feature definition, static inputs corresponding to user profile information are 88% (8/9) of total inputs, and dynamic inputs constituent the remaining 12% (1/9). Such feature definitions are not explicitly available for DLRM datasets that are available publicly. Thus, we assume a similar % split for them and divide the bottom MLPs in the static-dynamic input ratio of 87.5%-12.5%. The embedding tables are split into user-item embedding ratios of 75%-25%. Since there is no explicit feature description corresponding to each embedding in the dataset, we assume the first 75% to be user embeddings and the remaining 25% to be item embeddings to show the decomposability of the model. Furthermore, we also split the first two layers of the top MLP stack into the user and the item model; the last layer is not divided and is part of the combined top MLP.

The RM5 model and its dataset are more descriptive about feature definitions, so we can exactly decouple the user-history access and the candidate-specific accesses for the last two shared embedding tables. The decoupled user-history information is concatenated with the user_id embedding table entry to produce a user feature vector. Similarly, the decoupled candidate-specific information is the item feature vector. We split the first layer of the top MLP stack proportional to the user feature vector length and item feature vector length with a ratio of 60%-40%. The last two MLP layers are not divided and are part of the combined top MLP.

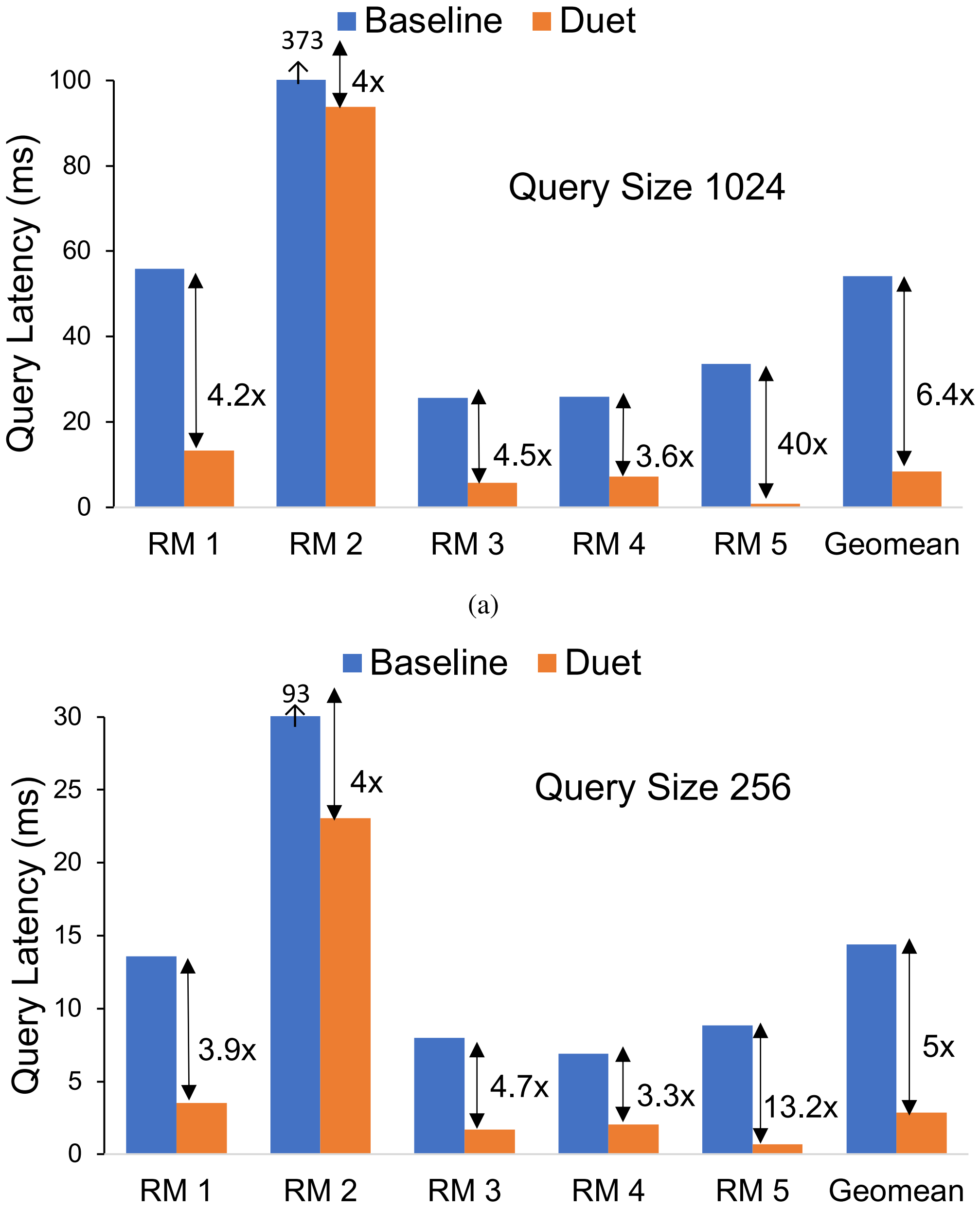
The energy estimates are extracted by means of the python RAPL library for the server computation; we also utilize battery state dumps comprising charge counters and voltage measurements for the mobile platform. The maximum data upload speed of 24 Mbps is obtained from a speed test on a mobile phone over the 5G network, and the energy estimate is obtained from a recent 5G characterization work [173].

5.4 Evaluation

In this section, we first present the query latency and energy benefits of Duet. We then demonstrate the accuracy impact and discuss the latency breakdown for individual components for different models. Next, we exhibit the benefits of the lightweight Duet architecture over Duet without hardware optimizations and discuss its overhead. We then show results for throughput over a wide range of batch sizes. Finally, we demonstrate the advantage of Duet with near-memory processing (NMP) support over prior NMP embedding approaches.

5.4.1 Performance

We demonstrate the reduction in query latency for query sizes of 1024 and 256 candidates in Figure 5.9. We observe that for a query size of 1024, the average latency reduces by 6.4× compared to the baseline. Similarly, for a query size of 256, we reduce the average query latency by 5×. Since our holistic approach resolves all three components of a recommendation model, we present consistent gains across all the models, which are bottlenecked on different components. The gains



(a)

(b)

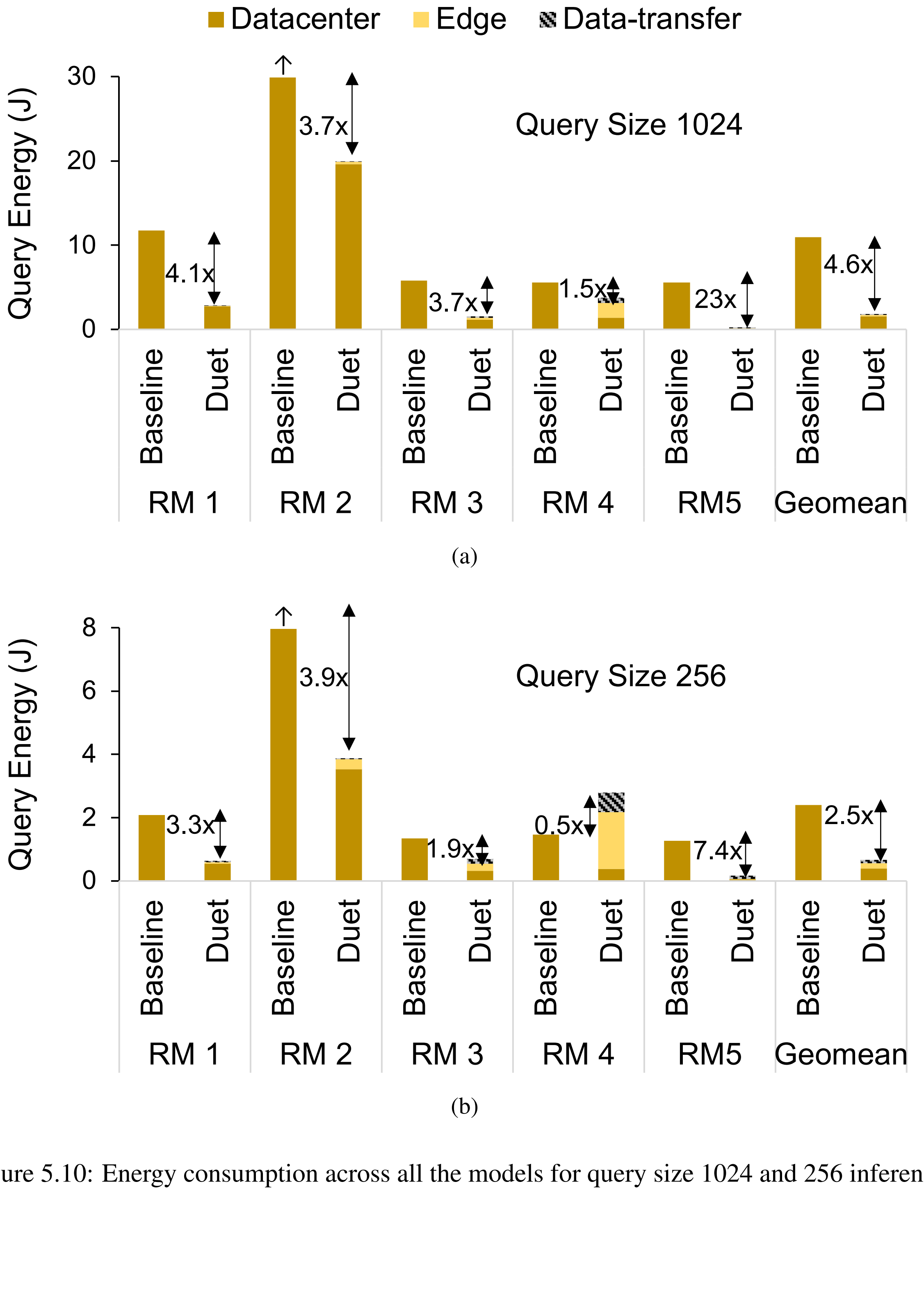
Figure 5.9: Query latency reduction across all the models for query size 1024 and 256 inferences.

are attributed to three factors: 1) decomposing the monolithic model into two smaller concurrent models, thus, overlapping the computations; 2) processing the user embeddings and dense bottom MLP once per query, which significantly reduces the DRAM access and MLP computation; and 3) optimizing the user model on edge with the lightweight Duet architecture. The architecture comprises an updated user-specific scratchpad, memoization for static bottom MLP, and lower qint8 precision for top MLP to reduce the computation and memory demands, which altogether eliminates(stops) the user model from becoming a bottleneck.

As observed in Figure 5.9, the RM1-RM3 models have higher gains over the RM4 model. This is because RM1-RM3 models compute a significant fraction of the dominating components, i.e., user embeddings and bottom MLP, once per query in an optimized fashion on the edge platform. However, RM4 computes the relatively heavy (item + combined) top MLP (>1MB) repeatedly for all the recommendation inferences on the datacenter, which consumes a considerable fraction of the total execution time. Meanwhile, RM5 has the highest speedup because of the following reasons. First, there is no separate item embedding for this model, which reduces the datacenter's item model size. Second, 199 of the 200 lookups for the two shared user/item embedding tables correspond to the user history, which is computed only once with edge platform optimizations. All inferences of the query reuse the edge computation, and each inference performs only one remaining lookup to rank the candidate. Thus, we reduce the number of embedding lookups from query_size×200 to query_size×1 for the embeddings. In a later section, we provide a detailed latency breakdown of various components on the edge and datacenter.

5.4.2 Energy

We exhibit the energy efficiency of our proposed solution, Duet, over baseline in Figure 5.10. The baseline constitutes only the datacenter energy consumption (brown). But our collaborative edge-cloud solution constitutes datacenter (brown), edge computation (yellow), and data-transfer (hashed) energy consumption. We observe an energy reduction of 4.6× and 2.5× for a query size of 1024 and 256 items, respectively. The average datacenter energy consumption decreases by an



(a)

(b)

Figure 5.10: Energy consumption across all the models for query size 1024 and 256 inferences.

average of 7× at the expense of an additional small fraction of energy consumption on edge. The datacenter's energy efficiency improves because Duet distributes the work in the user model at the edge and the item model at the datacenter, which overall reduces the amount of work completed on the datacenter. Moreover, our Duet architecture's energy-centric optimizations reduce the edge device's energy consumption. Below, we discuss the breakdown of energy consumption for a query size of 1024. Similar behavior is observed for a query size of 256, with deviation discussed separately.

For embedding-dominated RM1 and RM2 models, more than 97% of Duet's total energy is consumed in the datacenter, which computes item embeddings, item-specific top MLP, and combined top MLP. The remaining 3% of energy is consumed on the edge platform for user model processing and data transfer. For the bottom MLP-dominated RM3 model, the datacenter consumes 75% of the Duet energy, and the edge consumes the remaining 25%. The edge energy consumption increases from 5% to 25% because the heavy bottom MLP of RM3 is computed on the edge device. Nonetheless, because the bottom MLP is split into pre-computed static and dynamic bottom MLP, we lower the query energy by 3.7× by limiting the energy consumption to only dynamic bottom MLP.

Next, for the top MLP-dominated RM4 model, the energy spent by Duet on the datacenter and the edge is 36% and 64%, respectively. The edge device consumes more energy than the datacenter because, although quantized, a heavy user top MLP of 1.9 MB is computed on the edge device with 2MB of L3 cache. Quantization greatly reduces the user top MLP size from 7.5 MB to 1.9 MB, but the working set size with activations and other model components is still greater than the cache capacity. It does not completely eliminate DRAM overheads. However, compared to the fp32 model, the quantized qint8 model lowers the edge and the data-transfer energy to yield an overall query energy reduction of 1.3x.

An exception to the above-discussed results is the energy consumption of the RM4 model for a query size of 256 candidates. We find that the total energy consumption of Duet is greater than the baseline, which computes the entire model for 256 candidate inferences on the datacenter. This is

Model	Base (%) Accuracy	Duet w/o quant (%) Accuracy	Duet Accuracy (%)
RM1	78.71	78.68	78.64
RM2	78.67	78.66	78.62
RM3	78.75	78.71	78.69
RM5	68.74	68.97	68.97

Table 5.3: Accuracy comparison between Duet & baseline models

because the energy consumed by the heavy user top MLP at the edge platform, even though just once for a query, is higher than processing all the 256 inferences on the datacenter. Therefore, the energy consumption of the edge model could not be amortized across 256 inferences. However, for the query size of 1024, the user top MLP energy consumption is lower than 1024 inferences; thus, the edge platform energy was amortized across 1024 inferences, resulting in overall energy reduction.

We also demonstrate a significant energy reduction of 23× for the embedding-dominated RM5 model because the major fraction of the embedding lookups, attributed to the user history, are computed once on the edge platform in an optimized manner. Moreover, after Duet's distribution of work, the energy consumption becomes more balanced, with the datacenter and the edge platform consuming 46% and 54%, respectively.

5.4.3 Accuracy

We show the accuracy of our re-trained decomposed model for Duet and compare it with the baseline accuracy of the monolithic model in Table 5.3. We show that our Duet model (with bottom MLP split, user and item embedding split, and top MLP split) can achieve a comparable accuracy with an accuracy drop limited to $<=$ 0.03% across the models. Once the user top MLP adopts a quantized int8 precision, we achieve accuracy within 0.06% of the baseline accuracy, thus, still limiting the accuracy impact at a lower precision.

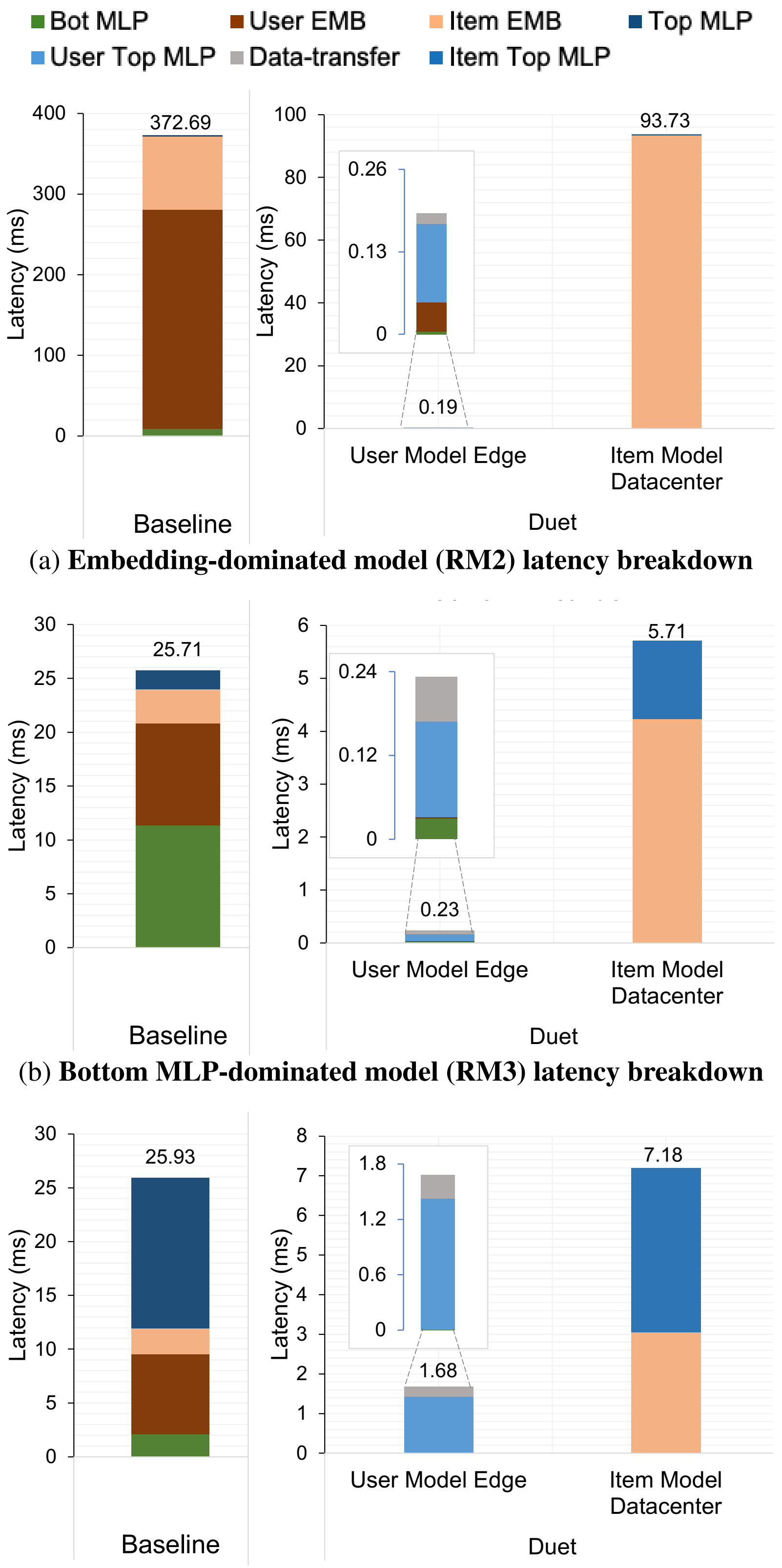
The benefit of quantization is more energy efficiency for the energy-constrained user device. We find that for a query size of 1024, the average query energy consumption for non-quantized Duet is 2.8×, which increases for quantized Duet to 4.6×. Meanwhile, quantization does not have any considerable impact on the end-to-end query latency because the edge user model is able to complete and produce its result well before the item model at the datacenter is ready to consume the edge result. Thus, the latency is bounded by the datacenter computation. Hence, we provide a trade-off where we can offer a more energy-efficient model with a minimal impact of 0.06% on the accuracy or reduce the accuracy impact to $<= 0.03\%$ without energy-specific optimization. In both cases, the accuracy impact is less than our threshold($< 0.1\%$).

5.4.4 Latency Breakdown

We have already discussed the overall query latency reduction for all the models and multiple query sizes in the prior section. This section presents the query latency breakdown for a query size of 1024 and describes, in detail, the time spent during execution on various components. We show the latency breakdown in Figure 5.11 for one of the embedding-dominated models (RM2), the bottom MLP-dominated model (RM3), and the top MLP-dominated model (RM4).

As presented in Figure 5.11a, the baseline RM2 model spends 73% of the time on user embeddings, 24% on item embeddings, 2% on the bottom MLP, and 1% on the top MLP. Decoupling the user and item embeddings and storing only user-specific embeddings on the on-chip scratchpad at the edge device has two benefits. First, we retrieve the contiguous user embeddings faster than random DRAM access on the datacenter. Second, we compute this significant fraction of user embeddings once for the query, reusing the computation across all the inferences of the query. The combined execution time of bottom and top MLP stacks is also reduced to <1% of total Duet execution time because of our hardware optimizations for MLP stacks. Eventually, the datacenter computation limits the overall performance after decomposing and offloading the lightweight user model to the edge device.

For the bottom MLP-dominated RM3 model, a major fraction of the execution time (44%) for the baseline is spent on processing the bottom MLP, as illustrated in Figure 5.11b. The remaining timing breakdown is 37% on user embeddings, 12% on item embeddings, and 7% on the top MLP. We optimize the bottom MLP and user embeddings by computing them once per query with our



(a) **Embedding-dominated model (RM2) latency breakdown**

(b) **Bottom MLP-dominated model (RM3) latency breakdown**

(c) **Top MLP-dominated model (RM4) latency breakdown**

Figure 5.11: Latency breakdown for RM2-4 models across all components for both the edge and the datacenter platforms.

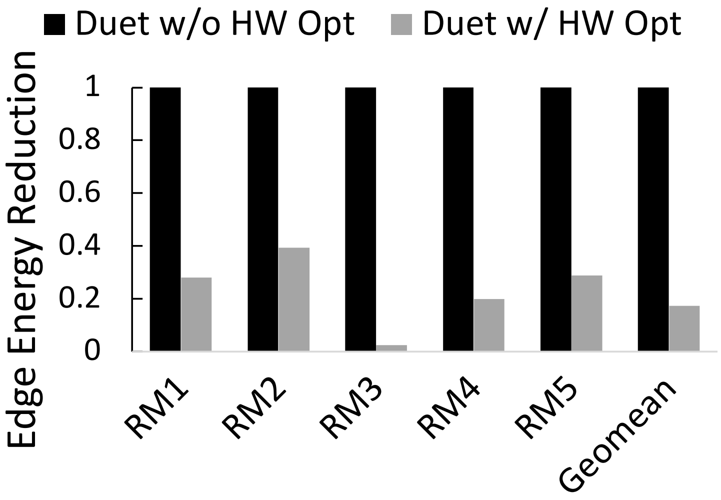


Figure 5.12: Impact of HW optimizations on edge device across all models.

Duet architecture on the edge platform, thus significantly reducing the individual component's execution time. The top MLP's execution time also decreases because the top MLP layers are decomposed, which decreases the computation in the datacenter. We also notice that the edge device spends the majority of time on the user model's top MLP. The two original bottlenecks are diminished because the hardware unit's memoization register stores static bottom MLP output, and its on-chip scratchpad optimizes user embeddings.

We exhibit in Figure 5.11c the execution time breakdown for the top MLP-dominated RM4 model. 54% of the time is spent on the top MLP, 29% on the user embeddings, 10% on the item embeddings, and 8% on the bottom MLP. Duet's architecture optimizes the top MLP time by splitting top MLP layers into user-specific, item-specific, and combined top MLP layers. The user top MLP on edge is computed once with qint8 precision and overlaps with computation on the datacenter. Moreover, splitting reduces the size of the datacenter's top MLP; thus, it completes all the 1024 inferences sooner than the baseline. Additionally, the user embedding and bottom MLP fractions are also reduced because of Duet's optimizations.

Note that an additional data-transfer time is added to Duet, but we limit it by reducing the payload size with the user top MLP processing and quantization.

5.4.5 Hardware Optimization Impact on Edge

Until now, we have presented results with all hardware optimizations enabled for the complete edge-cloud system. In this section, we study the energy impact on the edge device and show the benefits of hardware optimizations (Duet w/ HW Opt) over the software-based model decomposition (Duet w/o HW Opt). The Duet w/o HW Opt only decomposes the monolithic model into user and item models but does not enable hardware edge optimizations – hardware unit (user-specific scratchpad and memoization register) and qint8 precision. Though model decomposition reduces overall work through compute reuse, it adds work on the edge device due to the processing of the user model, which consumes considerable energy. As illustrated in Figure 5.12, hardware edge optimizations reduce this edge energy consumption of user models by 83% on average. Both the hardware unit and lower qint8 precision contribute toward the energy reduction of each model. However, significant gains for embedding-dominated RM1 and RM2 models are attributed to the user-specific scratchpad of the hardware unit. We also obtain a 97% energy reduction for the bottom-MLP dominated RM3 model because the compute-intensive static bottom MLP's output is memoized, significantly reducing the MLP workload during execution time. Top MLP-dominated RM4 model energy consumption is also reduced by edge-friendly lower (qint8) precision. Finally, all the models benefit from the reduction in data-transfer energy over 5G, owing to a smaller output (payload) size of quantized user top MLP. Overall, the hardware optimizations lead to only an 8% reduction in battery life compared to a 25% drop without any optimizations for a user query computed every minute. There is still a slight drop in battery life because of the additional compute added by the user model processing on the edge device.

We also compare the energy consumption of the end-to-end edge-cloud system for a query (size 1024) with hardware optimizations disabled and enabled. We observe that the average energy consumption is lowered by 1.8× without hardware architecture because of a reduced amount of work. However, the edge consumes a significant fraction of total energy. By enabling Duet's energy-efficient architecture for the edge device, we reduce the system's query energy by 4.6×.

In conclusion, model decomposition decreases overall work, resulting in query energy reduc-

Model	Scratchpad Size (KBs)	Scratchpad Area (mm^2)
RM1	60	0.020
RM2	720	0.238
RM3	17.5	0.006
RM4	17.5	0.006
RM5	28.2	0.009

Table 5.4: Scratchpad Overhead

tion. Nevertheless, it also computes the user model on edge, which consumes significant energy at the energy-limited device. Our proposed Duet architecture reduces the edge energy consumption; thus, it is extremely important for the viability of model decomposition.

5.4.6 Hardware Unit Silicon Overheads

Table 5.4 presents the on-chip scratchpad size required for storing user-specific embeddings by the different models. These are derived from a recent SRAM design in 10nm technology node [93]. The location tracker is synthesized separately using a 15nm commercial library at a clock frequency of 2.8Ghz. The post-synthesis timing is just 0.33ns@0.66mW and occupies $250um^2$. Thus, our location tracker is an extremely lightweight unit. The majority area of the hardware unit is occupied by the scratchpad. Embedding heavy RM2 requires the largest scratchpad space of 720 KB, corresponding to $0.24mm^2$ of silicon area. In contrast, all the other models require only a few tens of KB of scratchpad and $<0.1mm^2$ area. Thus, Duet is a high-performance and energy-efficient solution with negligible area overheads.

5.4.7 Sensitivity to Batch Size

Figure 5.13 illustrates throughput improvement over a wide range of batch sizes from 8-1024 for embedding-dominated model RM2, bottom MLP-dominated model RM3, and top MLP-dominated model RM4. In this work, we assume a single node server computes all the items of the query. However, a query can be composed of multiple batches that are distributed by the front-end server across multiple servers. We demonstrate that Duet increases the average throughput by 3×. Models

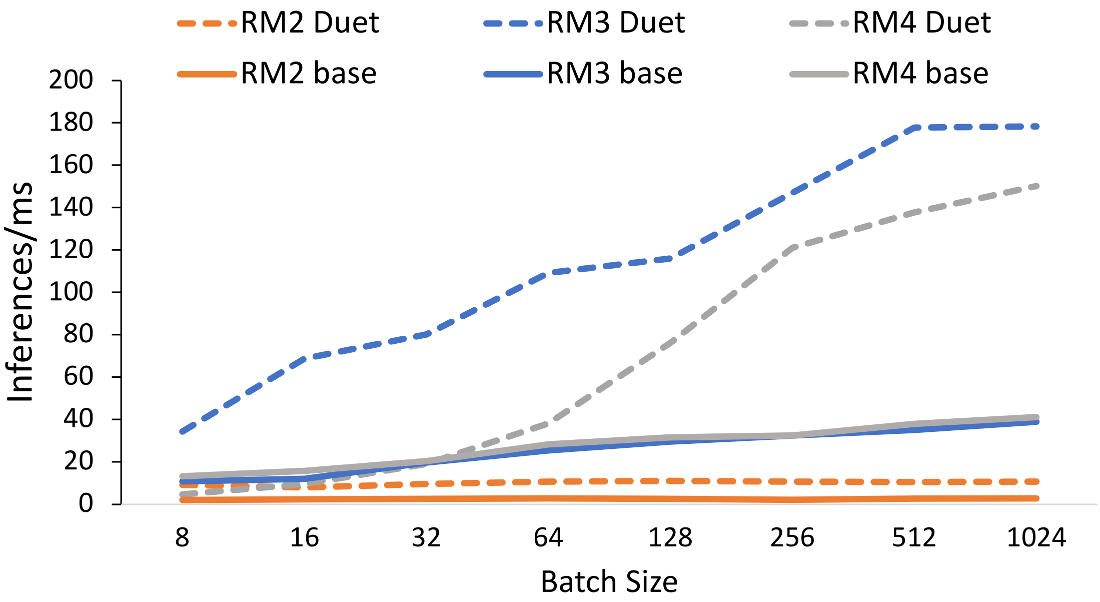


Figure 5.13: Throughput improvement for multiple models over a wide range of batch sizes

RM2-3 provide consistent throughput gains for all batch sizes. However, throughput decreases for the RM4 model at smaller batch sizes of 8, 16, and 32. This behavior is because the user model's execution time at the edge device is higher than the total time taken to process all the items of a batch, sized 8-32, in the datacenter; hence, edge model computation becomes a bottleneck. The RM4 user model's edge latency could not be amortized across 8-32 inferences of a batch because the heavy user top MLP (1.9MB) of RM4 cannot be completely cached, leading to higher latency than RM2-3 user edge models. RM2-3 models are embedding and bottom MLP heavy; the pre-computed static bottom MLP and an on-chip scratchpad make these models lightweight, even for small batch sizes.

5.4.8 Comparison to NMP solutions

We also compare Duet with two recent state-of-the-art near memory processing (NMP) based techniques: RecNMP [133] with an average 10× embedding speedup and TRiM [184] with an average 39× speedup (3.9x reported speedup over RecNMP). The NMP solutions are targeted to resolve the embedding bottlenecks. Notably, as a secondary impact, they also improve the performance of MLP layers because of lowered cache evictions of MLP weights and activations,

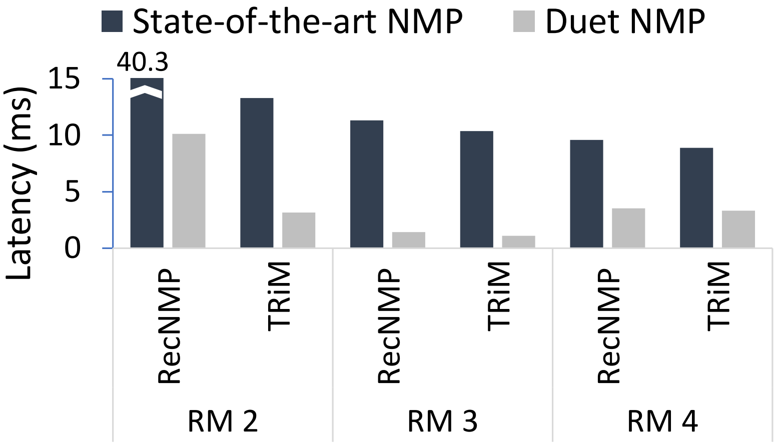


Figure 5.14: Comparison of NMP-enabled Duet with state-of-the-art NMP solutions.

which otherwise are replaced by embedding vectors. The NMP techniques are complementary to our solution; therefore, we also utilize these techniques for our item model computation at the datacenter for a fair comparison. Our NMP-enabled Duet solution has two parallel running models. The user model at the edge is by Duet's architecture without NMP support, and the item model at the datacenter is computed with NMP enabled for the item embeddings. To support NMP for our framework, we scale the item embeddings by the embedding operation speedups reported by prior works and determine the MLP speedups by performing the MLPs standalone on the server without any interruption from embeddings, thus, eliminating eviction impacts.

In Figure 5.14, we compare the NMP-enabled Duet with the two solutions – RecNMP and TRiM – for embedding-dominated model RM2, bottom MLP-dominated model RM3, and top MLP-dominated model RM4. Duet-NMP decreases the average query latency by 4.18× and 4.2× over RecNMP and TRiM because the distribution of models reduces the amount of work that is completed at the datacenter compared to computing the entire model on the datacenter. Thus, at the datacenter, Duet-NMP completes a lesser amount of work faster than the state-of-the-art NMP solutions.

We also show improvements for MLP-dominated RM3 and RM4 models and for high NMP embedding speedups (thus, practically eliminating the embedding bottleneck) because of the decomposition of the MLP stack on the edge and the datacenter. A high speedup of 8.7× is observed for the RM3 model, as we memoize static bottom MLP output and, thus, eliminate that computation. For the top MLP-dominated RM4 model, the average speedup is 2.7×, which is comparatively smaller than RM3. This is because the remaining item top MLP at the datacenter consumes a considerable execution time by repeatedly performing all the inferences of the query. Hence, overall, our holistic approach that individually optimizes all the components of the recommendation model can provide speedups over state-of-the-art NMP solutions across models with different bottlenecks.

5.5 Discussions

Scalability: Duet's scalability is driven by the size of user-specific embedding tables, which, compared to entire embedding tables (10+ GB), are small because they store only a fixed length of the user history. While for the current models considered in this work, the size of user history is limited to $<$ 1MB. However, if the user history becomes too large for future models, we may have to employ intelligent tiered solutions by partitioning user-specific embedding tables into scratchpad and DRAM. To reduce DRAM overheads, we can leverage near-memory processing for user-embedding operations on DRAM at edge devices. Moreover, new technologies like dense STT-RAM and embedded NVMs can accommodate larger user-embedding tables.

Training Cost: We train the decomposed model from scratch for Duet, as discussed in Section 5.2.5. Although the model is trained once offline in the datacenter, still training the model takes 10+ hours. Therefore, it is important to reduce the training time. Many recent works [48, 49, 147, 196] have focused on accelerating the training of the recommendation models, which can be adopted by Duet to reduce the training overhead. The other option is to amortize the training cost by serving thousands of queries over a long period of time. To our advantage, the weights and embedding values of the decomposed model are generic and not fine-tuned for individual users; thus, it can be offloaded to any user device, serving queries from all users at any point in time.

5.6 Conclusion

Recommendation models process multiple candidates to recommend only a few to the user. They also exhibit non-uniformity in their inputs, which can be categorized as user inputs and item inputs. This work explores the opportunity to decouple the two inputs and operate on user-related inputs at the local edge device, thus enhancing data privacy. We present Duet, a novel collaborative edge-cloud recommendation system, which decomposes the giant monolithic model into two smaller concurrent models – a user model on edge and an item model on the datacenter – that come together to deliver final recommendations. The user model is computed once per query by our new energy-efficient Duet architecture on the resource-constrained edge device, and its output is reused by all the candidates computed by the item model on the datacenter for the query. We demonstrate that our proposed lightweight Duet architecture reduces query latency by an average of 6.4× and lowers average energy consumption by 4.6×. Until now, we have proposed hardware-software co-design approaches which modify the application based on the hardware present in the edge device. In the next chapter, we will present a new hardware design and efficiently map popular mobile applications on the proposed hardware architecture.

CHAPTER 6

Interconnect Architecture for System-in-Package Based Low-Cost Edge Platforms

As discussed in previous chapters, the goal of this dissertation is to improve the consumer experience for edge platforms. In this chapter, we provide a performance and energy-efficient low-cost mobile architecture. Mobile/Smartphones represent a major fraction of the space of edge devices. At present, smartphones have a Systems-on-Chip (SoC) at their heart for all computational purposes. While SoCs are becoming more powerful to fulfill the computational demands of emerging applications, they are also simultaneously getting expensive to develop. SoCs require huge initial capital investments because of their monolithic nature, shift to expensive technology nodes, the inclusion of more custom IP blocks, and lower yields. This is driving up the retail cost of smartphones for users. To replace sophisticated, expensive SoCs, we need a cheaper alternative with comparable or better performance and energy efficiency to serve user demands for powerful and reasonably priced smartphones. In this chapter, we explore the potential of the emerging 2.5D-based System-In-Package (SiP) architectures to lower these costs while simultaneously improving the performance and energy efficiency of edge applications.

In a SiP, individual IPs are manufactured separately as *chiplets*. These small chiplets are then integrated on an interposer substrate using a low-cost assembly step, avoiding many other expensive and mandatory steps required to integrate SoCs monolithically. The modular nature of SiP has many other benefits, like cheap reuse of IPs across multiple domains, higher yields, heterogeneous integration of multiple technology nodes, and shorter time to market. Clearly, SiP has the potential

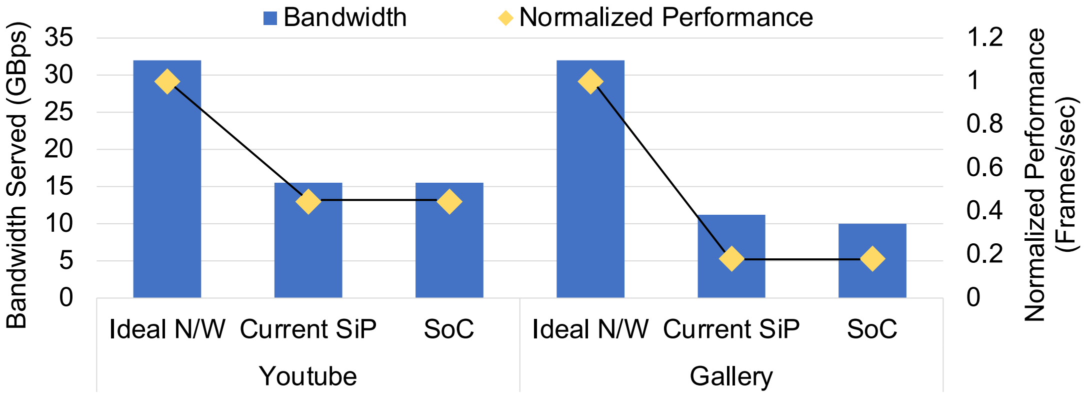


Figure 6.1: Bandwidth served by an ideal network (with infinite bandwidth), recent SiP, and SoC for YouTube and Gallery applications.

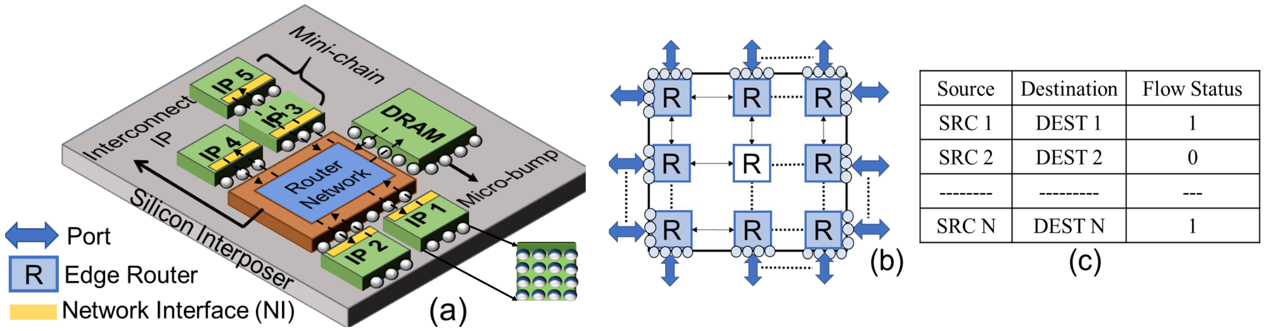
to reduce the startup cost of indigenous heterogeneous systems like SoC and IoT systems.

In this chapter, we propose Neksus – an interconnect architecture for SiP designs based on 2.5D stacking on an interposer substrate. Neksus includes a general and flexible interconnect layer to support modular "plug-and-play" of chiplets while mitigating the area overheads of micro-bumps connecting to/from the interposer layer. The design comprises a hub-interconnect chiplet that connects to IP chiplets via mini-chains, paired with a Network Interface (NI) and SerDes module within each IP chiplet. Besides decreasing micro-bump area costs, our mini-chains are tailored to support common data-flow patterns of applications, which are typical of mobile and IoT platforms. Data-intensive applications, such as video streaming and audio-playback, consist of recurring patterns in which data frames flow sequentially (*i.e.*, in a pipelined manner) through several IPs, each writing and reading intermediate (and final) results to and from memory. In this work, we show that such flow patterns can benefit from the mini-chain structure, which enables direct IP-to-IP communication with no intermediate data buffering in memory. Finally, we develop a protocol shell within the NI to support compatibility with existing SoC protocols, offering a low barrier for porting current infrastructures to our proposed architecture.

In our evaluation, we show that mini-chains improve application performance by 28% with 31% energy savings in a SiP system. We also study the deployment of mini-chains in SoC systems and show that it can improve performance by 28% with 36% energy savings.

6.1 Motivation

SiP-based designs have an immense potential to reduce the startup cost of developing a new heterogeneous chip by allowing cheap reuse of IP. However, for the emerging 2.5D-based SiP framework, a flexible interconnect solution that addresses the design constraints of SiPs is still lacking. Hence, in this work, we present a **holistic interconnect architecture** for SiP systems. Our approach is governed by three key design principles based on SiP characteristics. First, to **enable plug-and-play** of IP chiplets, the interconnect architecture should require minimal changes to the chiplets' designs, and the network layer should be largely **decoupled** from the IPs. Second, we strive to **minimize the number of expensive micro-bumps**, which facilitate the electrical connection between chiplets and interposer layer. They are expensive area resources as their size does not scale at the same rate as the feature size of transistors. For instance, a 256-bit bidirectional port with 40μm pitch micro-bumps occupies $\sim$0.41mm^2, while a bitcoin chiplet [136], at the 16nm technology node, occupies only $\sim$0.23mm^2. This gap between chiplets size and micro-bump port size will increase as we move to lower technology nodes for better performance. There is a need to reduce the number of micro-bumps to avoid the extra silicon cost of accommodating more micro-bumps. Lastly, the network layer should **leverage high speed/bandwidth interposer link to support bandwidth-intensive SoC applications**. Recent SiP [131] interconnects or conventional mesh based SoC are unable to serve the bandwidth requirements of mobile applications, which can improve performance from increased bandwidth provided by an ideal network, with infinite bandwidth, as shown in Figure 6.1. Towards this end, we present Neksus, a solution that features an exclusive interconnect chiplet and a network interface to satisfy the SiP design requirements and leverage interposer properties to support real-world applications.



Source	Destination	Flow Status
SRC 1	DEST 1	1
SRC 2	DEST 2	0
--------	---------	---
SRC N	DEST N	1

Figure 6.2: Neksus, the proposed SiP architecture (a) - Interconnect IP (b) - Flow control table at edge routers (c).

6.2 Neksus Architecture

6.2.1 Interconnect Chiplet

We propose a dedicated interconnect chiplet acting as a central hub for two main reasons. *First*, a dedicated chiplet supports modular "plug-and-play" integration since it minimizes the required changes to existing IP chiplets. *Second*, in SiP, it enables heterogeneity by allowing IP chiplets from multiple sources to connect directly to the interconnect chiplet. Figure 6.2 shows the SiP integration model proposed in this work. The interconnect chiplet connects to other chiplets via dedicated ports. Signals from each port travel through sender IP micro-bumps, the interposer metal layers, and receiver IP micro-bumps to reach their destinations. An internal router network is integrated within the interconnect chiplet, supporting any topology specified at design time.

Passive v/s Active. Although this work presents a passive interposer solution connecting to a dedicated interconnect chiplet, a similar approach could be devised for an active interposer architecture. We chose a passive interposer because of its more immediate commercial feasibility [37, 190], lower cost [207], and potential of choosing upcoming organic [185]; glass substrates as interposers [208, 139], which are cheaper and provide better performance than silicon. We discuss in detail about active interposer and its pros and cons in Section 6.6.3.

6.2.2 Mini-Chains to Reduce Overhead

Micro-bumps are the interface between chiplets and the network channels through the interposer. Signals from each port travel through sender IP micro-bumps, the interposer metal layers, and receiver IP micro-bumps. The number of micro-bumps used is proportional to the number of ports in the interconnect chiplet. We introduce the concept of *mini-chains* to reduce it. Mini-chains allow ports to be shared across multiple chiplets by chaining several IPs together in a unidirectional ring. We choose a unidirectional ring over a bidirectional ring or multidrop bus in order to minimize micro-bump overheads and support a completely passive interposer. Each unidirectional mini-chain is connected to a port on the interconnect chiplet. Therefore, a single port of the interconnect chiplet multiplexes over all the IP chiplets within a mini-chain, as shown in Figure 6.2. These chains can support data flows only among non-overlapping source-destination pairs at any point in time. Chaining eliminates the need for a dedicated port for each IP chiplet on the interconnect chiplet and hence helps keep its size smaller, making the architecture scalable as the number of chiplets grows. Most importantly, using mini-chains provides high-bandwidth wires connecting multiple IPs within the chain directly, without the overheads or limitations of a router.

The size and composition of mini-chains can be governed by the applications' behavior, with IPs heavily interacting with each other placed on the same chain. Furthermore, the data flow pattern can guide the relative placement of IP chiplets within a mini-chain. For example, IPs sharing a producer-consumer relationship should be placed at a one-hop distance in the mini-chain.

For IP chiplets to communicate, packets are first routed over the source IP's mini-chain, then through the router network in the interconnect chiplet, and then over the destination IP's mini-chain. In a typical on-chip network, routers are used to arbitrate between packets transmitted over a channel. In contrast, there are no routers within a mini-chain; IP chiplets on a chain have simple network interfaces. The interconnect chiplet's *edge router*, which connects externally to the mini-chain, is responsible for arbitrating among the IPs on the same chain. We adopt a low-cost hand-shaking mechanism to enable this arbitration.

Hand-Shaking Mechanism. To enable inter-IP communication within a mini-chain, additional

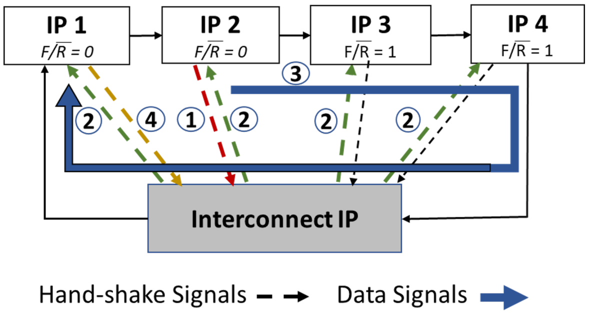


Figure 6.3: Hand-Shaking protocol: The sequence of steps indicates how to setup and tear down a data transfer from IP2 to IP1.

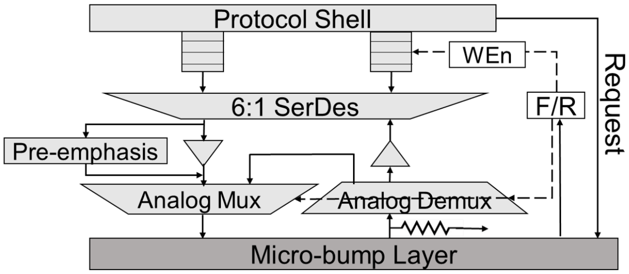


Figure 6.4: Network interface for IP and interconnect chiplets.

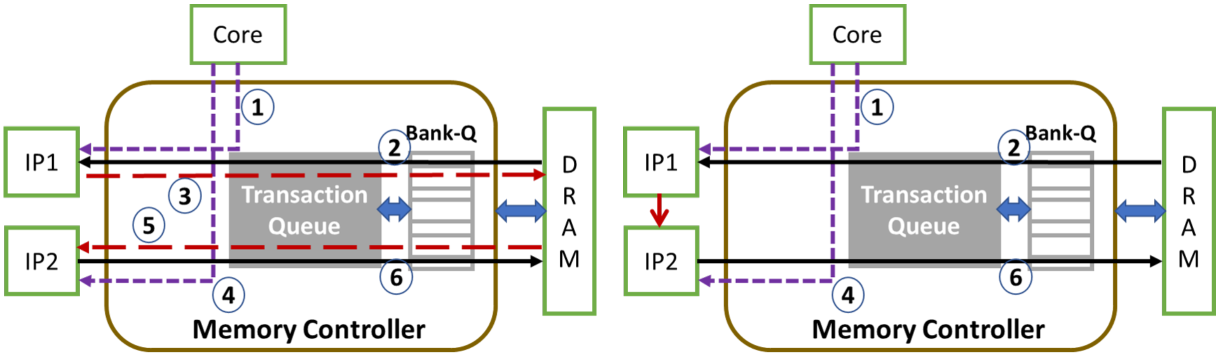
logic bits, Forward/Receive (F/R), are required at the chiplet interface. These bits decide whether to forward the incoming data to neighbors or accept it for processing. Each chiplet sends a request to the edge router in the interconnect chiplet over dedicated 1-bit control links (dotted lines in Figure 6.3). The interconnect IP keeps small flow control tables, as shown in Figure 6.2(c), to track the availability of data-links for each mini-chain. Each mini-chain has a dedicated flow control table with 12 entries (representing each possible src-dest pair in a 4-IP mini-chain) tracking src-dest pair flow statuses. Consider, for example, a scenario in which IP2 wants to communicate with IP1, also illustrated in Figure 6.3. First, a request signal (1) is sent over the handshake link to the interconnect IP. Once the src-dest pair entry is available, the interconnect IP sends an ack signal (2) and sets the F/R bits at each IP. Specifically, the forward (F) bit is set to 1 at intermediate IPs and to 0 at the receiver IP1. After the F/R bits are set, the data flow (3) begins. When the tail flit reaches its destination, a reset signal (4) is sent to update the flow status in the lookup table. Setting the flow control bits requires only one extra cycle since it uses a fast, direct, 1-bit control link to each IP chiplet and only a small table lookup. Note that the reset by the tail flit can be carried out concurrently with de-serialization and, therefore, does not incur an extra cycle of delay. The broadcasting of information is done as in NoC, where the packet is duplicated at multiple points in the network. If the broadcast bit is set, the packet is replicated and sent to the next IP or router in the network. In another scenario, if the intermediate data is needed at a later instant in time, it will also be directed to the memory module that stores a valid copy. There is no extra overhead incurred to keep a valid copy as compared to the baseline. In the baseline, two network access - memory writes and memory read by the next IP - is required. Neksus also needs two network access - sending to the next chained IP and writing to memory - in case the intermediate data needs to be stored for later use.

6.2.3 Application-Level Chaining

The proposed mini-chains can also efficiently support the data flow patterns of applications typically run on hand-held devices. Previous work [225] has investigated the inefficiency of the data flow patterns in these applications. In particular, read/write accesses to DRAM are unnecessary when the memory simply serves as a buffer, as illustrated in Figure 6.5(a). In the figure, arrows 3 and 5 show the data-path for reads/writes to the memory. Upon receiving a request from the core (1), IP1 reads data from DRAM (2). Once the processing is complete, it writes the result to DRAM (3). Once IP2 receives a request from the core (4), it reads the data (5) stored in memory by IP1 and starts processing it, eventually writing back to DRAM (6). Note that although this example uses only two IPs, many more may be involved in the pipelined data-flow, depending on the application. For example, in a Skype call, video and audio flows can be pipelined in various chains: camera and video-encoder; video-decoder and display IP; microphone and audio-encoder; audio-decoder and speaker IP.

Bypassing: To alleviate this problem, it is possible to bypass memory as in [225] by reading data from the memory controller's transaction queue and bank queues or by forwarding the data from a scratchpad. While the performance would improve, the network bottleneck would remain unresolved, as much of the network traffic must still traverse the congested links at the memory controller or scratchpad.

Subframing: Unfortunately, this IP-to-IP data reuse can require substantial buffering if there is a delay between data production by the upstream IP and its consumption by the downstream IP. Buffering requirements can be on the order of a single frame, which would be infeasible. Sub-



(a) Data transferred via DRAM

(b) Mini-chain data transfer

Figure 6.5: Data flow solutions for IP-to-IP communication.

framing [225] can be employed to mitigate the large buffering needs; it facilitates the processing of data at a much finer granularity using small data units rather than complete frames. With the original approach, the second IP is only invoked after the first IP completes the processing of one frame and has written the result back to DRAM. But with sub-framing, once a small chunk of a frame (*i.e.*, a sub-frame) is processed, it can be passed immediately to the downstream IP for processing. This technique is viable because applications naturally tend to work at fairly small granularities, which can be used as sub-frame units: tiling for rendering, slicing for video codecs, and independent audio frames for large audio files.

IP-to-IP Chaining: Neksus is particularly well-suited to facilitate efficient data-flow. For example, the data flow depicted by arrows 3 and 5 in Figure 6.5(a) can be replaced by a network path connecting IP1 and IP2 directly, as in Figure 6.5(b), thus bypassing DRAM and eliminating the network bottleneck. This is because IP1 (upstream IP) and IP2 (downstream IP) share a producer-consumer relationship, where IP2 uses data produced by IP1 as input for processing. This direct IP-to-IP data flow can be efficiently mapped onto the mini-chains in Neksus. Direct data flow from an upstream IP to a downstream IP can take advantage of high-bandwidth links through the interposer without paying the penalty of going through routers. Since current IP designs can support similar data-rates [38], buffering a couple of sub-frames is sufficient. Our evaluation uses 1KB sub-frames and buffering capacity of 2KB for each IP. Mapping application chains to the interconnect's mini-chains is expected to improve performance because it enables high-bandwidth

IP-to-IP communication and enables data pipelining among multiple IPs. Energy savings result from a reduction in IP active cycles due to faster computation because of decreased memory stalls and a reduction in memory access, which, in turn, reduces network energy.

6.2.4 Network Interface

A *Network Interface* (NI) is placed in all Neksus' IP chiplets, as shown in Figure 6.4. The NI has a protocol shell, a couple of incoming/outgoing flit queues, a SerDes interface, a flow-control bit (F/R) that determines whether the data has to be received by the current IP or forwarded to the downstream IP, and finally, a layer of micro-bumps. The data flow is pipelined through three stages: serialization, link traversal, and de-serialization. The signal passes through SerDes only at the endpoints of the transmission to minimize the latency penalty, while it bypasses it at the intermediate nodes (interconnect and chiplets in a mini-chain) through a multiplexer. Since high-data-rate signals will be traveling through the interposer wires, these must use differential, or ground-reference signaling [185]. For this purpose, our design uses analog multiplexers. The maximum signal loss for 4 hops of the multiplexer+interposer track, as found from the SPICE simulation using a 45nm technology node library [175], is 8dB (using the interposer and micro-bump RLC models from [132]). To overcome this signal loss, an optional pre-emphasis block at the transmitter can be used, which can easily compensate for a 10db loss in signal, as discussed in prior work [185].

SerDes to reduce Microbumps: A complementary approach to reduce the number of micro-bumps at *each port* for *all* chiplet is by leveraging serial communication via a Serializer/ Deserializer (SerDes) module at the NI. Essentially, SerDes is a high-frequency multiplexer capable of sending multiple bits through a single micro-bump in a short window of time, enabling the same number of data bits to be sent through fewer micro-bumps within a comparable timeframe. In our architecture, a 6:1 SerDes is used to reduce the number of micro-bumps by a factor of six (reduced 256 micro-bumps per port to $\sim$44). SerDes adds one cycle of extra latency at each end of the transmission. We designed and synthesized a mux-based SerDes on an IBM 45nm technology node with Synopsys Design Compiler and found that we can drive a 6:1 mux at 10 GHz (0.1ns delay), which is well within our 0.5ns requirement. The synthesized SerDes module takes an area of 42μm^2 and consumes 0.34mW of power.

6.2.5 Protocol Compatibility

To make Neksus compatible with current IP designs, which may support one of several popular industry or customized protocols, there is a need for protocol migration. NI's protocol shell handles this migration. Neksus is based on packetized network flow-control, which is yet to be supported by existing SoC protocols to the best of our knowledge. Hence, there is a need to build an interface for standardizing existing SoC protocols by packetizing the signal at the protocol shell. To demonstrate an example protocol shell, we choose AXI4 [34], which is a widely supported protocol for on-chip communication in SoC products. AXI4 is a point-to-point socket protocol and can be applied for communication among multiple IPs via an interconnect.

We modeled the proposed protocol shell in SystemVerilog, with an IBM 45nm technology node. The design synthesizes at z 2GHz frequency and has a 330μm^2 area.

Interfacing with DRAM. An exception to the above NI architecture is the DRAM chiplets. Most DRAM modules used in hand-held devices use an LPDDR interface[205]. Due to the complex timing requirements in the LPDDR standard, it is not efficient to packetize the LPDDR protocol. Instead, we use a memory controller IP chiplet that communicates with other chiplets using a packetized format and communicates with DRAM directly through LPDDR signals.

6.2.6 Neksus for System-on-Chip

Neksus addresses the design constraints of interposer-based System-in-Package, in addition to exploiting its unique properties. Note, however, that Neksus is not restricted to SiP platforms. While SoC platforms do not suffer from the same technology constraints as SiP, SoCs can still enjoy the benefits Neksus provides to typical mobile applications through mini-chains. Thus, Neksus can be repurposed to SoC as follows. Since SoCs are not limited by micro-bumps, the SerDes in the

Network Interface can be removed. However, since Neksus is a packet-switched network, the NI must still host the proposed protocol shell along with multiplexer logic for forwarding. Note that Neksus on an SoC platform can support more wires, as it is not limited by micro-bump pitch, and more wires can be placed at an SoC port. So, even though individual SoC wires are slower than properly buffered fat interposer wires, which results in lower bandwidth per wire, the SoC bandwidth would end up equivalent to that of a SiP interposer by having more channels. For this work, we assume the port bandwidth of SiP and SoC are the same for a fair comparison.

6.3 Cost Model

6.3.1 Cost Model

To understand *quantitatively* the impact that a SiP architecture may have on startup investment, we developed a cost model for two platforms: a SiP and an SoC. The proposed cost model breaks down the total cost into two major components: 1) Non-Recurring Expenses (NRE) and 2) Recurring Expenses (RE). The model is based on ASICs' cost breakdown from [136, 83] and interposer cost analysis from [153]. The cost analysis modifies the cost parameters from those works to account for low-cost reuse of hard chiplets and provides benefits to SoC design wherever possible to be conservative.

6.3.2 Non-Recurring Expenses

Non-recurring Expenses (NRE) refer to the one-time lump-sum capital required for a startup. This cost is amortized over time as the product enters the mass production stage. However, it is the major contributor to the initial investment capital. It takes into account various factors, such as mask cost, design labor & design tools cost, verification cost, and IP procurement/licensing cost. Table 6.1 lists the components that apply to SiP and SoC, which are also described below.

Mask cost: Mask cost is one of the major contributors to NRE cost. At the 28nm technology node,

	SoC	SiP
NRE	Complete System Mask Cost Complete System Design Cost New IP Verification Cost Reused IP License Cost	New IP Mask Cost New IP Design Cost New IP Verification Cost
RE	Complete System Wafer and Test cost	New IP, Interposer, Reused Chiplets Wafer and test Costs

Table 6.1: SiP and SoC design cost components

Parameters	New IP SoC	New IP SiP	Reused IP SoC	Reused IP SiP
BE labor cost/gate($)	0.131	0.164	0.131	0.164
BE tool cost/gate($)	0.331	0.414	0.331	0.414
FE labor cost/gate($)	0.065	0.065	0.019	0.00
FE tool cost/gate($)	0.026	0.026	0.008	0.00
Gate Count for 4mm^2 chiplet (K)	4917	4917	4917	0
Top level module integration gates(K)	15	15	15	15

Table 6.2: NRE parameters

it can be as high as $2,250k and is increasing rapidly due to double patterning for 16nm and lower. Owing to its monolithic nature, SoC requires the development of a new mask for the complete system on changing even a single component, whereas, in SiP, only new components need mask development.

Design cost: Design cost captures the development cost from scratch for the new IPs on both platforms. We took the largest ASIC designed by prior work [136] and listed the front-end (FE) and back-end (BE) labor and tool cost per gate in Table 6.2, for new as well as reused IPs. Reused IPs make the difference between both platforms. For SiP, reused IPs will be taken off the shelf and, therefore, will not require any front-end (FE) and back-end (BE) design efforts for chiplets. Additional back-end tools and engineering effort may be required owing to 2.5D stacking over an interposer. Thus, to take this aspect into account, we increased the BE cost per gate for SiP by 25% over SoC, which manifests in the form of extra person-months and CAD-months for IPs in SiP. Nonetheless, for SoC, reused IPs will be delivered in the form of soft cores. Thus, the SoC design requires both FE and BE costs for each IP. However, the FE engineering efforts required will be less than that of designing IPs from scratch. This observation is captured by reducing the FE costs per gate for reused IPs in SoC by 70%, as also reported in Table 6.2. Module/Chiplet integration cost should be added to back-end engineering cost for both SiP and SoC. Therefore, the top-level gates, as reported by [136], accounting for I/Os and routers, are added for each IP.

Licensing cost: Soft cores procurement will include licensing costs for each reused IP. The SiP-based design avoids this cost. From [136], IP license for standard cells, SRAM is approximately \$100K.

Verification cost: There are two components of verification: individual IP verification and system-level verification for IP integration. We assume to have reused soft cores in SoC or off-the-shelf chiplets pre-verified by the vendor. Only new IPs will introduce the extra verification cost for integrating with existing components. This cost will be the same for SiP and SoC. According to Foster [83], verification consumes $\sim$60% of total turnaround time. We add this cost for new IPs to the design cost to obtain the total product development cost.

We recognize that hard macros can be used instead of soft cores. However, using hard macros will restrict design flexibility, reusability, and portability. For example, new licenses will be needed when migrating a design to a new technology node, eliminating the benefits of reusability. Moreover, mask cost and licensing cost of hard macros will still exist, which are significant, as discussed above. Thus, we do not explore this option further.

6.3.3 Recurring Expenses

RE refers to the cost customers pay for the manufactured chip. Thus, it includes only the silicon cost, which can be due to the wafer, interposer, and off-the-shelf chiplet costs, as described below.

Wafer cost: SoC will incur wafer cost for the complete monolithic system, whereas SiP will include wafer cost for only new IPs, as listed in Table 6.1. The wafer parameters used are shown in Table 6.3.

Interposer cost: SiP will also include the interposer cost. Each SiP platform is assumed to use an interposer die of the size of the complete system with an extra 10% area overhead. The cost of

	Wafer(28nm)	Interposer (passive 65nm)
Diameter	300mm	300mm
Cost per wafer ($)	7600	1500
Stacking Cost/chiplet ($)	-	0.05
Test cost Active silicon area ($/mm^2)	0.01	0.01
Test cost Interposer ($)	-	0.27
Test cost microbump probing/chiplet ($)	-	0.05

Table 6.3: RE parameters

passive interposers, as reported by prior work on a cost analysis for interposers [153], is shown in Table 6.3, which includes the TSV creation, BEOL, RDL, micro-bump, and C4 bump processing, at $\sim$\$1500 per wafer and with production throughput of 50k/month.

Off-the-shelf chiplet & Bonding cost: SiP will also include off-the-shelf chiplet costs for reused IPs. This is simply the wafer cost for each chiplet since they are assumed to be mass-produced. An extra cost is also added to include the stacking/bonding cost for each chiplet. Currently, the bond yield of stacking is 99% [149], which can be increased by adding a few redundant micro-bumps by deploying approaches similar to those used to improve TSV stacking yield [129, 160, 114] from 15% up to 99.99%. We compute that the 99% 2.5D bond yield can be increased to 99.99%, with just 0.1% redundant micro-bumps and a pessimistic 10% area overhead for extra logic embedded in each chiplet. Another work [80] proposes using ECC at each chipset to increase the assembly yield for microbump with interposer to almost 100% with a small power overhead of ECC.

Testing Cost: Table 6.3 also reports the testing cost [212], scaled for our system, incurred by active dies, interposer die, and microbump probing. SoC will just incur the testing cost of only the active monolithic chip, whereas SiP will incur all three costs.

6.3.4 Yield Modeling

The yield factor represents the fraction of good dies over total manufactured dies. We assume 300mm diameter wafers. Defect density (Do) of 2000 *defects/mm*^{2} for the wafer, 200 *defects/mm*^{2} for passive interposer, $Frac_{crit}$ of 0.9 and 0.01, for chiplets and passive interposer respectively, which determines critical area *Acrit* for transistors and α of 1.5 are used in the yield model, which

Size Interconnect Chiplet	8mm^2
Size Chiplet	2mm×2mm
#Chiplets/#Ports	64/16
Interposer/Chiplet Technology Node	65nm/28nm
IP-to-IP Channel length	2mm
Interposer Delay	42 ps/mm
SerDes Delay	1cycle each
Router frequency	2Ghz
Max B/W Achieved	32GB/s
Chaining factor	4
SerDes Operating Frequency	10Ghz

Table 6.4: Neksus design parameters

is based on the classic equation provided by [73]:

$$Yield = (1 + \frac{DoAcrit}{\alpha})^{-\alpha} \tag{6.1}$$

6.4 Evaluation Methodology

6.4.1 Design Parameters

Design parameters used for our Neksus architecture are listed in Table 6.4. A passive interposer, with an extra area overhead of 10% over the complete system size, is used. The interposer wire (at 65nm) delay is estimated to be 42ps/mm [57], and a channel size of 2mm is used. An interconnect chiplet of size 8mm^2 with 16 ports is used, with micro-bumps spread over a small area at each port. Table 6.4 also lists the design parameters of the SerDes.

Note that for our cost model, instead of a minimum-sized chiplet, we assume an average-sized chiplet, which may not suffer from micro-bump constraints. However, this choice makes our cost analysis fairer, even though our architecture seeks to alleviate constraints faced by *all* chiplet sizes.

Core	1 way, 1.8GHz
Memory	2GB/300Mhz
Mem-ctrl TransQ	64 entries;
Mem-ctrl Bank-Q	24 entries
Chiplet Clk	500 Mhz
Video/Img size	4K/1080p
Audio size	16Kb

Table 6.5: GemDroid Settings

6.4.2 Performance Model

We evaluated the Neksus architecture using two approaches. The first is a design exploration approach, where different topologies are examined for Neksus' router network in the interconnect IP. We also evaluate Neksus for best-case and worst-case traffic patterns. This topology exploration is carried out in the cycle-accurate Booksim simulator [123]. All routers use a three-stage microarchitecture. We use simple deterministic routing algorithms, finite input buffering, and virtual-channel flow control. The network assumes a packet size of 64B (512 bits) for memory requests/replies and a subframe size of 1KB for direct IP-IP communication, 4 VCs per port, with a per-VC buffer size of 4 flits. Neksus uses mini-chains of 4 chiplets, and thus it implements a 16-port network, whereas the SoC includes a 64-port router network. Because of the unidirectional nature of mini-chains, only non-overlapping data flows are supported at a time. For an interposer wire delay (at 65nm) of 42ps/mm [57] (i.e., 94ps for unbuffered 2mm channel), and a maximum distance between chiplets of 10mm, two IPs within a mini-chain can transfer data at a 470ps latency accounting for 5 hops through intermediate IP. Signals are buffered at the network interface of each intermediate die which restricts the maximum unbuffered distance in Neksus to channel length (i.e., 2mm here). This falls within 1 clock cycle at the network frequency of 2GHz. Therefore, the unidirectional nature of a ring does not impact the performance of the system. Apart from this, an extra cycle is spent to set the Forward/Receive bits for each packet. We measure both latency and bandwidth in our evaluation to compare different topologies.

The second approach analyzes the performance of the architecture on application traces from Youtube, Video-Record, PhotoCapture, Gallery, Audio-Record, and Skype. For this approach, we

YouTube	VD-DC; AD-SND
Video-Record	CAM-VE-MMC; MIC-AE-MMC
Photo-Capture	CAM-IMG-DC; CAM-IMG-MMC
Audio-Record	MIC-AE-MMC
Gallery	MMC-IMG-DC
Skype	VD-DC; CAM-VE; MIC-AE; AD-SND

Table 6.6: Application chains

Router link (28nm)	0.07 pJ/mm-bit
Power per VC	2.5 mW
SoC Router crossbar (8×8)	12.57 pJ/transaction
SiP crossbar (16×16)	18.6 pJ/transaction
SerDes power	8.14 mW
Interposer link (65nm)	0.23 pJ/mm-bit
DRAM [65] [172]	10 pJ/bit
IP Buffer Read/Write [169]	0.042 pJ/bit
IP power	150 mW

Table 6.7: Power model parameters

collected multiple application traces from an android emulator. We then run them through the Gemdroid framework [72] that has 11 IPs, along with a core and memory module, which was modified to support the proposed architecture. Gemdroid is integrated with the proposed Neksus architecture modeled in Booksim to support closed-loop simulations. GemDroid settings are listed in Table 6.5. Performance is measured as frames completed within a per-application deadline. In Table 6.6, we report the chiplet chains exercised by the applications.

6.4.3 Power Model

The power demands of Neksus can be partitioned into three components: 1) Network power, 2) DRAM power, and 3) Active IP power. Network power captures power spent on routers, the SerDes, and the interposer. Routers crossbar, buffer, and channel power were assessed using SPICE modeling for a 28nm industrial process. The crossbars have a matrix architecture and embedded arbitration at cross-points [195, 197]. DRAM power depends on the number of memory accesses. The power for the IP units is calculated based on active cycles and peak power. Table 6.7 lists the power parameters used for the modeling. The peak power of IPs is estimated to be 150mW.

6.5 Results

6.5.1 Cost Analysis

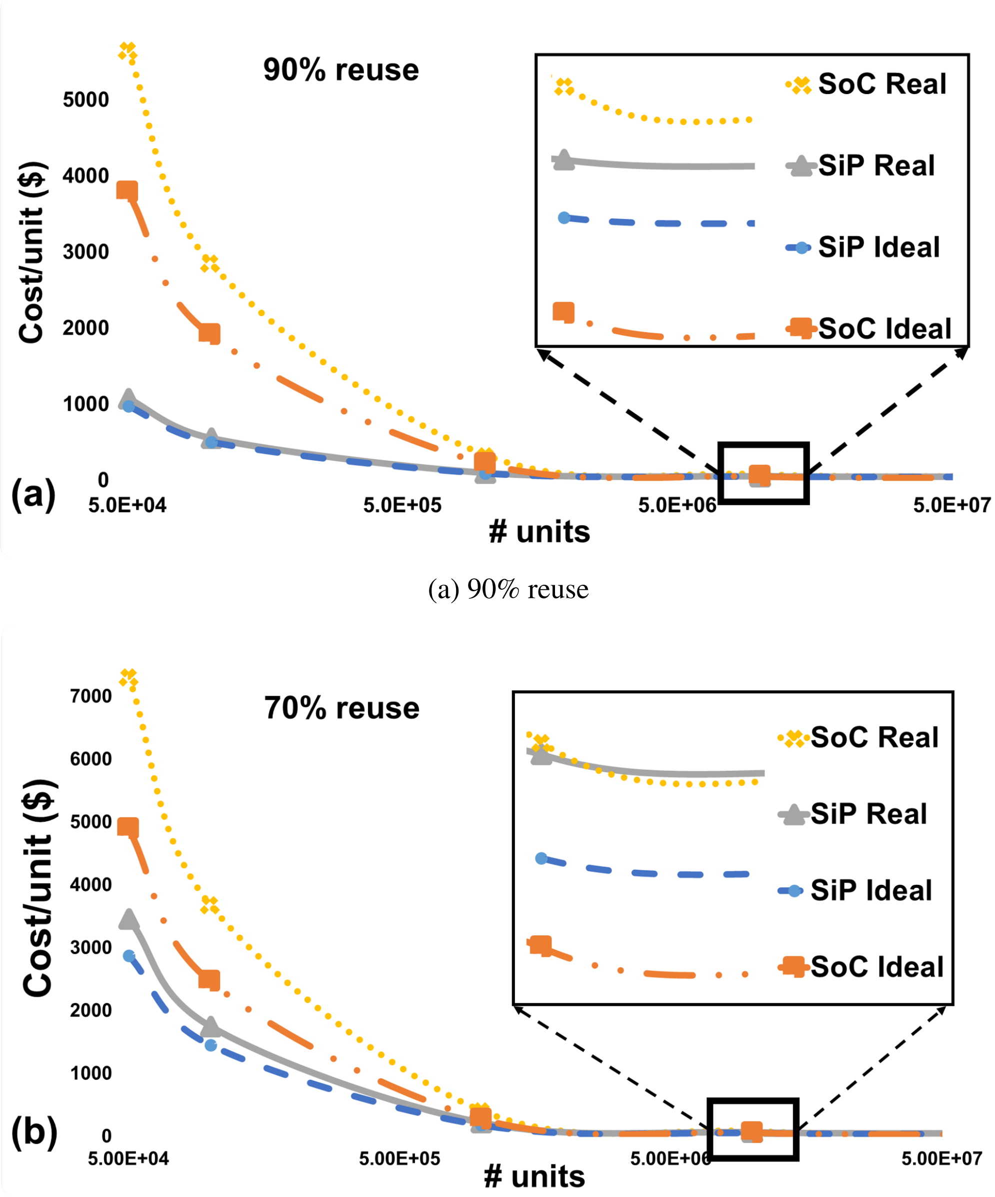
Cost Model Validation:

ARM projections [226] report the cost of developing a constant area (100 mm^2) chip for different technology nodes and across numbers of units manufactured. The projections account for the NRE cost of design and mask. We validated our cost model using the design and mask cost parameters from Section 6.3.2, and the transistor counts for two commercially available chips: (1) Apple A7 with die area of 102 mm^2 and one billion transistors [35] at 28nm; (2) Intel Core 2 Duo Wolfdale with a die area of 107 mm^2 and 411 million transistors [33] at 45nm. Assuming 4-8 transistors per gate, based on the technology node, we found that our cost model is accurate within 4-6% for one million and 50 thousand units. Hence, this validates our cost model's design and mask NRE factor.

For high-volume production (beyond 100 million units) where NRE is amortized, and only silicon wafer cost remains, the cost diverges from the projections because of the difference in wafer cost estimates used by the two approaches. Note that we take a more pessimistic silicon wafer cost; thus, we show a worst-case analysis at the mass production stage.

Our cost model captures the SiP reuse characteristic across domains in the form of a reuse factor. The reuse factor is defined as the fraction of IPs in the SiP that is bought off-the-shelf as silicon chiplets assembled on the interposer. Since most IPs remain unmodified as they are ported to new platforms or are bought as soft cores for development, we use reuse factors of 90% and 70% in our models, which aligns with the 80% reuse factor reported by Darpachips [9]. Using these two reuse factors, we study the cost per system for SoC and SiP as the number of systems manufactured grows.

Ideal Yield: As shown in Figures 6.6(a) and 6.6(b), with perfect yield, given our reuse factors,



(a) 90% reuse

(b) 70% reuse

Figure 6.6: Cost with perfect and real yield for (a) 90% and (b) 70% reuse factors.

we note that the startup cost for the SiP is lower than that of the SoC design by $3.9\times$ and $1.7\times$, respectively. This result is due to three major factors: mask, design, and licensing cost. For SiP, new masks are created for only new IPs, whereas for SoC, a new mask has to be prepared for the complete system. The design cost of SoCs is high since it requires extra engineering effort for reusable soft cores, whereas the design cost is low for SiPs since it requires design effort only for integration. Similarly, licensing costs are required for SoCs but not for SiPs. With a 90% reuse, the cost gap is higher. This is because the mask and engineering requirement cost decreases more for SiP than for SoC; SoC also suffers an increase in licensing cost. As the number of parts in production increases, the one-time cost is distributed across millions of parts. Therefore, at mass production, only silicon cost is left. Towards the tail of the graph, as shown in the inset of Figure 6.6(a),(b), we observe that SoC is cheaper than SiP because SiP incurs the extra cost of the interposer, stacking, and testing. Note that the above analysis assumes perfect yield and is thus not realistic. If defects are taken into consideration, a realistic yield will determine the total number of good dies that can be carved out of a wafer.

Realistic Yield: It is common knowledge that small chiplets have a higher yield than big monolithic designs. From that model, using a defect density of 2,000 defects per mm^2, a 256 mm^2 SoC has a 66.9% yield, compared to 93% yield for a SiP on taking into account new chiplets, interposer, and bonding yields. It is interesting to note the impact that yield has on cost. *Accounting for yields, with 90% and 70% reuse factors, we observe that startup costs can be reduced by* $5.2\times$ *and* $2.3\times$*, respectively*, as shown in Figures 6.6(a) and 6.6(b), since the SiP delivers a higher yield than the SoC, it amortizes its cost over more parts. Therefore, cost savings increase more for the SiP. Moreover, at the tail of the plot in Figures 6.6(a), (b) (see inset), the cost for the SiP is slightly lower or similar to the SoC at billions of parts. This is because the extra cost overhead for SiP due to the interposer, stacking, and testing are compensated by the SoC's lower yield.

6.5.2 Network Performance

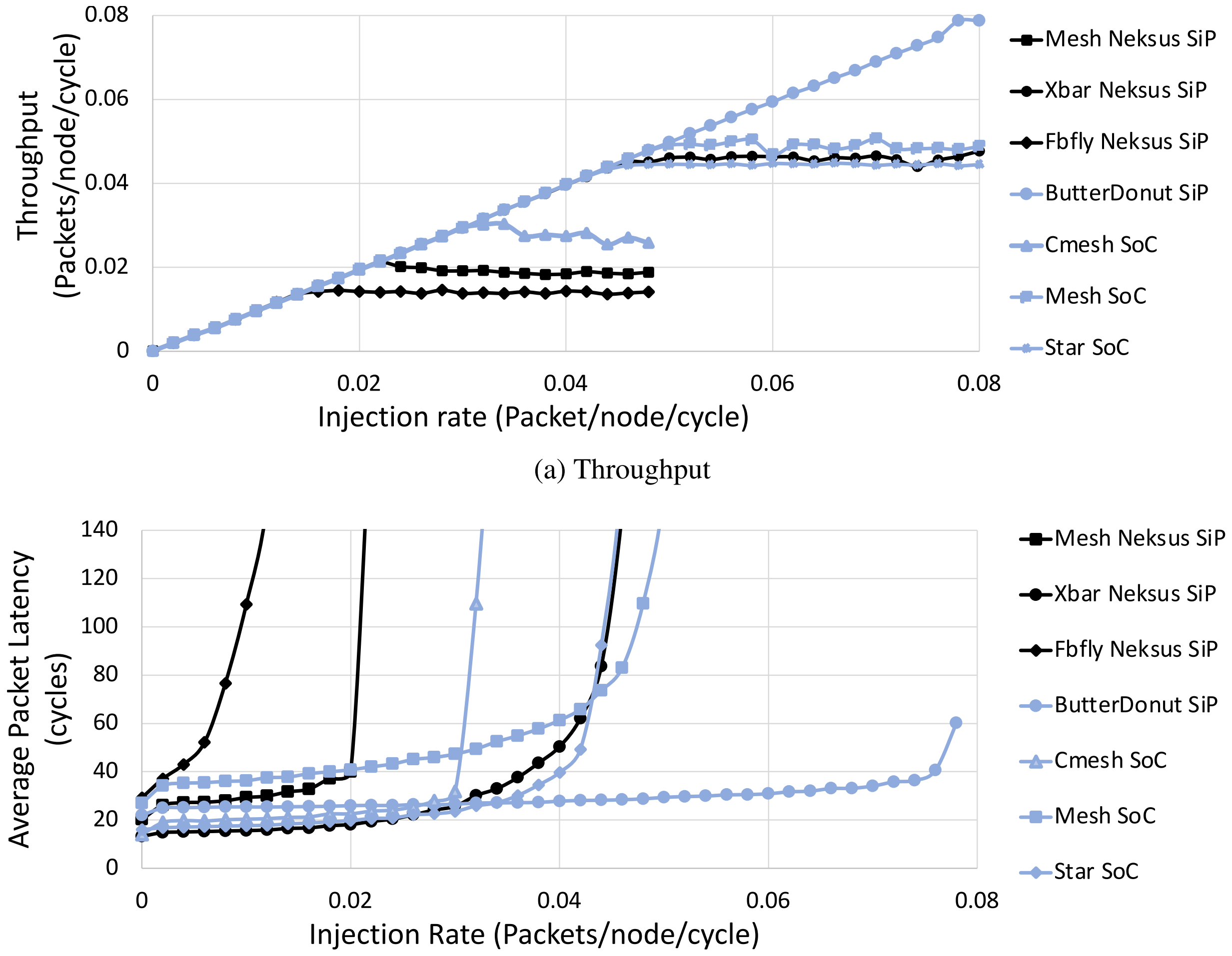
Neksus Topology Exploration:

In this section, we examine three interconnect topologies within the Neksus' interconnect chiplet: a 4×4 mesh, a 4-ary 3-flat 2-dimensional flattened butterfly, and a 16×16 crossbar. Here mesh is similar to a grid structure, with each router connecting to three neighboring routers and an input/output node; the crossbar connects n inputs to m outputs without intermediate stages in a single hop; and flattened butterfly is one of the most widely used high radix topology where each row of the network of a conventional butterfly (k-ary n-fly) is combined or flattened into a single router [138]. We then compare these three topologies against the following Network-on-Chip topologies: an 8×8 mesh, a Star topology with a local router degree of 4, connecting to a single global router, and a 4×4 concentrated mesh with a degree of 4. We also compare against the misaligned ButterDonut (BD) topology proposed by prior work [131] for interposer-based SiP.

As shown in Figure 6.7, latency and bandwidth results for uniform random traffic indicate that the crossbar topology is the best suited for the central hub characteristic of the interconnect chiplet. Crossbar has 57% lower latency and is within 6% bandwidth of mesh; has 12% lower latency and is within 3% bandwidth of Star; has 23% lower latency and 52% more bandwidth than concentrated mesh. Among the interconnect chiplet topologies considered, the crossbar provides the performance closest to the NoC topologies. However, for Uniform Random traffic patterns, the crossbar has 41% lower bandwidth and 38% lower latency than the SiP ButterDonut topology. This result is in sync with prior work, which reports that Butterdonut is the best fit for the uniform random traffic patterns found in multicore systems. This study also represents worst-case traffic evaluation since uniform random traffic represents an all-to-all communication where each node sends a packet to every other node in the system, which comes with a lot of overlapping data flows. Even though mini-chains in Neksus can support only non-overlapping data-flows at any instant of time, which penalizes Neksus for overlapping data-flows, Neksus with a crossbar is able to perform on par with other SoC topologies which can support overlapping data-flows simultaneously.

Benefits of Mini-chain:

Since our proposed Neksus architecture supports mini-chains, we also created a synthetic chained traffic, where IP1 communicates with IP2, followed by IP2 communicating with IP3, and IP3



(a) Throughput

(b) Average Packet Latency

Figure 6.7: Interconnects performance comparisons: (a) throughput and (b) packet latency with uniform random traffic.

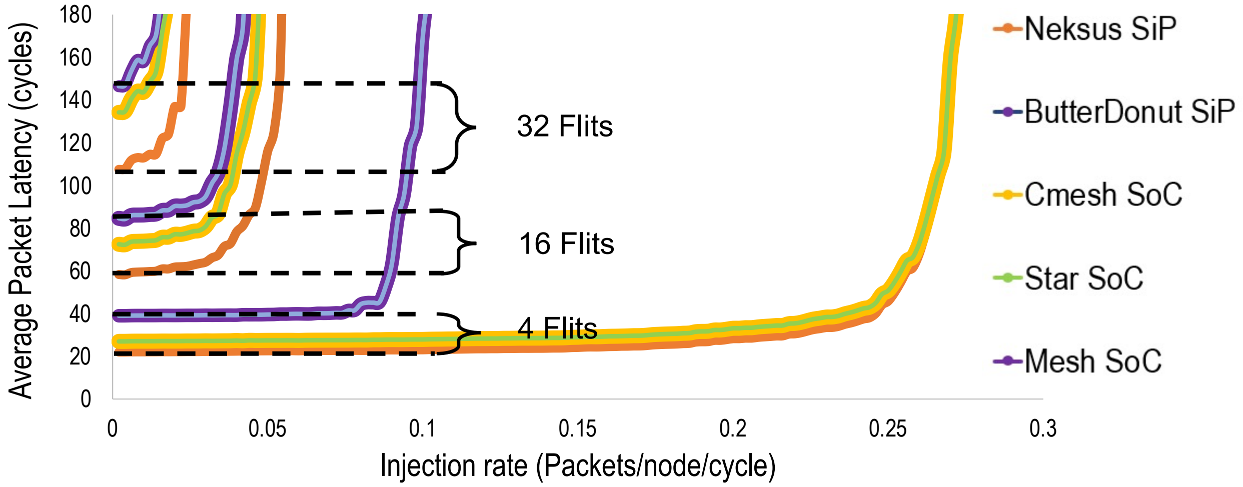


Figure 6.8: Chained traffic performance for Packet size = 32 flits, Packet size = 16 flits, and Packet size = 4 flits

with IP4, with all communicating IPs localized in the same mini-chain. This pattern is used to assess the performance benefits of mini-chains in isolation over different packet sizes. As shown in Figure 6.8, the mini-chain within the Neksus is the best solution over the mesh, concentrated mesh, butterdonut, and Star for synthetic chained traffic. Star and concentrated mesh present the same performance. This is because they share the same underlying network architecture utilized by the synthetic local traffic, which is a single router connected to four IPs. Similarly, ButterDonut uses mesh topology locally within each chiplet, and thus, mesh and BD have the same performance. We observe that for the competitive SoC & SiP topologies—cmesh and ButterDonut—Neksus's latency advantage increases to 24% and 32 %, respectively, as the packet size increases to 32 flits. This is because of the high-bandwidth direct IP-IP communication supported by the mini-chain, in contrast with router-centered communication. Also, the delay required for setting up the communication for each packet is amortized as packet sizes increase.

Therefore, an important observation is that, for traffic patterns following pipelined data-flow patterns, which are common in heterogeneous systems, the Neksus architecture with a crossbar interconnect and mini-chain provides better performance than regular homogeneous NoC architectures.

6.5.3 Application Evaluation for Neksus

We studied 6 applications listed in Table 6.6 on two SiP architectures—Neksus and ButterDonut (BD). For this evaluation, we examined three dataflow configurations: 1) direct IP-to-IP communication, where application and software stack are co-designed so that IPs talk directly with each other rather than going through memory (*Network-IP-to-IP*), 2) via-buffer communication, where memory requests are forwarded from caches/buffers at the memory Controller chiplet (*Network-via-buf*), and 3) via-memory communication, where a producer IP writes everything back to memory and the consumer reads it from memory (*Network-via-mem*).

Bandwidth Evaluation:

Figure 6.9 shows the bandwidth attained by the different configurations. For any individual IP, Neksus with IP-IP communication serves a peak bandwidth of $\sim$31GBps, compared to Neksus-via-buf, which only serves up to $\sim$12 GBps. This is because multiple IPs are actively trying to read from the memory controller chiplet; thus, network congestion from the memory controller chiplet link becomes a bottleneck. The Neksus-via-mem configuration suffers from memory bottleneck and is only able to utilize up to $\sim$11Gbps of available bandwidth. Neksus with IP-IP communication is also better than both, ButterDonut with buffering and Butterdonut with IP-to-IP communication configurations, which serves a peak bandwidth of $\sim$25GBps. This is because Neksus can support multiple non-overlapping data flows in each mini-chain, whereas regular router-based topologies are bounded by the router microarchitecture, which can transfer only one data packet at a point in time. We also observe that audio applications such as audio-record require less network bandwidth because of their small frame sizes.

Performance Evaluation: Figure 6.10 plots frames completed within a deadline. The bandwidth served by the different configurations determines their performance. We observe that Neksus with IP-IP communication reports 2.9x and 4x, on average, better performance over the Neksus-via-buf and Neksus-via-mem, respectively. Furthermore, Neksus with IP-IP communication has better performance than both the BD configurations by 16% on average, peaking at 28% over BD with IP-to-IP communication.

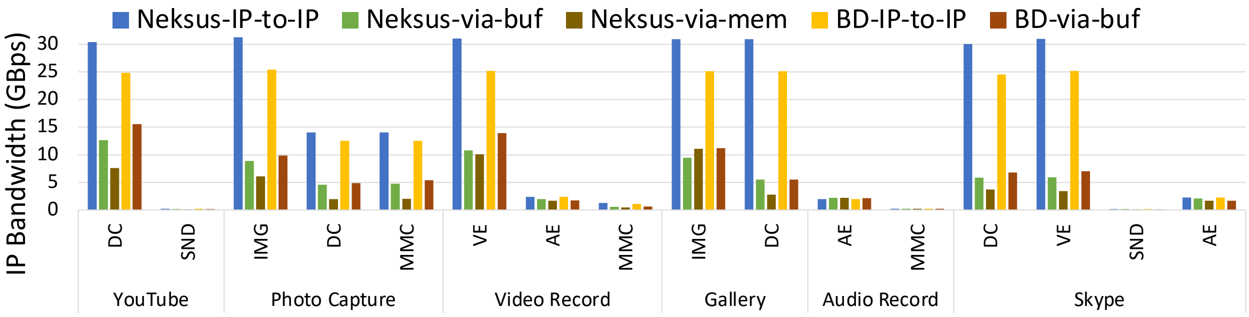


Figure 6.9: IP data transfer bandwidth for mobile applications.

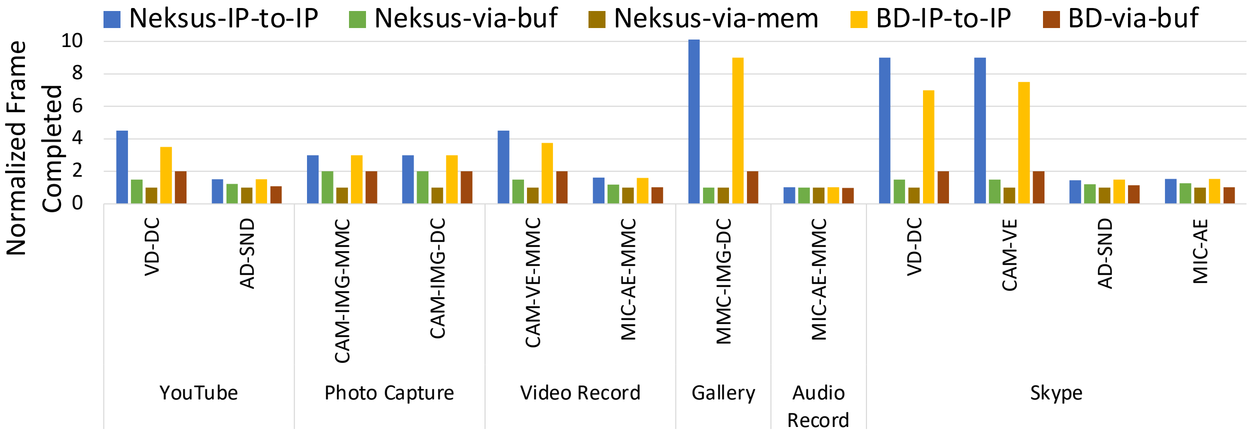


Figure 6.10: Frame completion rates for mobile applications.

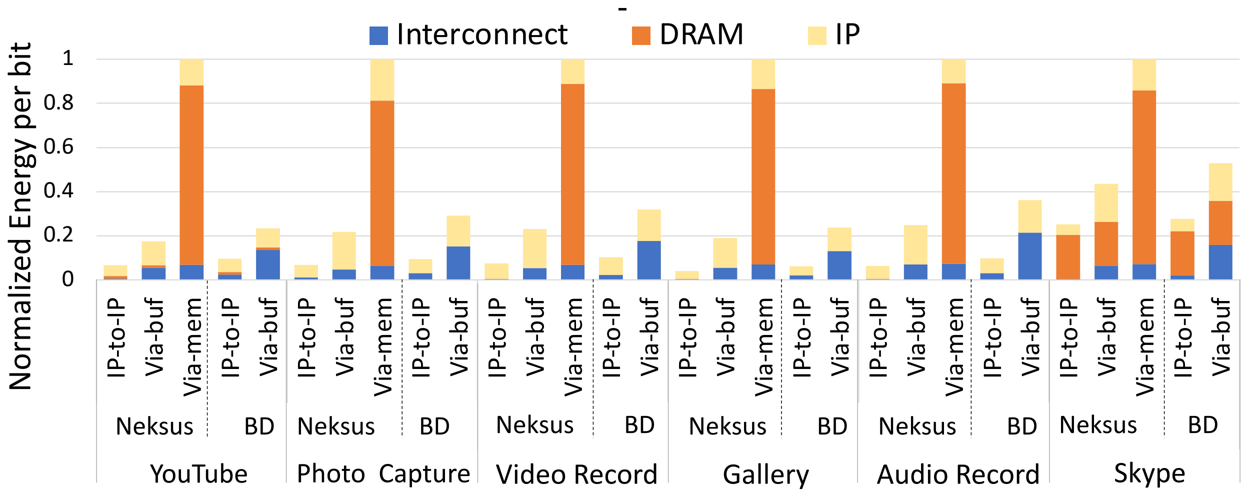


Figure 6.11: Energy per bit transferred spent by interconnect, storage, and IP units, over a range of architectural solutions.

For audio applications like audio-record, Neksus provides no significant improvements. This is because audio uses small frames, requiring less network bandwidth. However, in application scenarios where audio chains run concurrently with video chains, the memory controller has to service large amounts of data. Thus, significant improvements can be seen with Neksus for audio chains as well as with IP-to-IP communication configuration. Most of the performance gains are seen with image/video applications, which work on high data rates and high-resolution frames. This is because they require significant data movement and are bandwidth intensive.

Energy Evaluation:

Figure 6.11 shows the average energy per bit. We find that Neksus with IP-IP communication is more energy efficient by 60%, and 88%, on average, over the Neksus with buffering and via-memory configurations, respectively. Similarly, Neksus consumes less energy than ButterDonut with buffering and ButterDonut with direct communication by an average of 70% and 31%, respectively. The network-related energy improvements in Neksus with IP-IP communication are because we avoid router hops along with their buffer reads/writes, replacing them with the interposer. Energy improvements of Neksus with IP-to-IP communication over Neksus-via-mem configuration can be attributed to eliminating redundant read/write accesses to DRAM where DRAM is accessed only as a source/sink for IP chains, but not as a buffer.

	Neksus SiP	Neksus SoC	SoC w/ buf	SoC
Normalized EDP	0.047	0.043	0.28	1
Normalized Cost	0.189	1	1	1

Table 6.8: Cost and efficiency tradeoff.

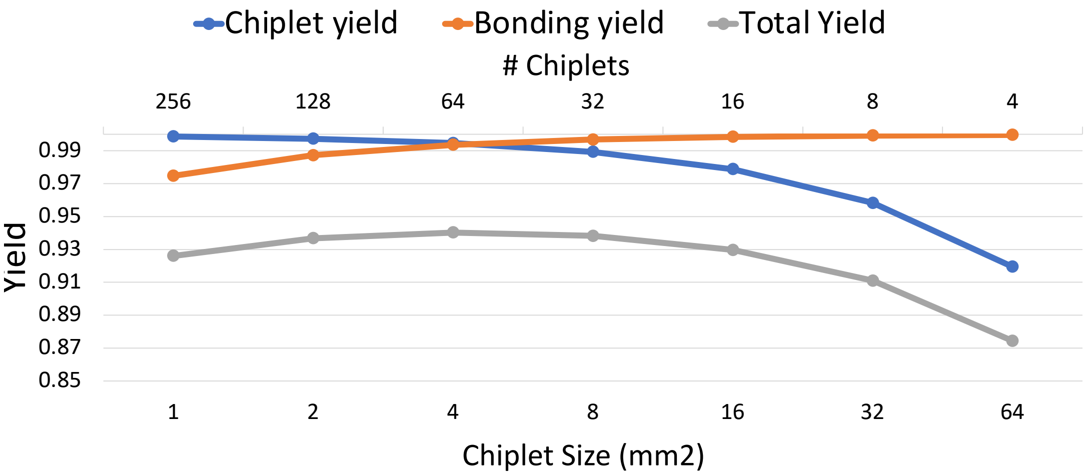


Figure 6.12: Chiplet, bonding, and total system yield for various chiplet granularity.

6.5.4 Cost-Efficiency Tradeoff for SiP and SoC

As discussed in §6.2.6, the Neksus architecture can be deployed in an SoC platform as an optimized interconnect for mobile applications. Neksus for SoC provides similar bandwidth as for SiP but does not incur SerDes-related overhead (extra power consumption). However, in using SoC, we lose all the benefits of SiP—modularity, heterogeneity, and reusability, along with reduced startup cost. Table 6.8 shows that Neksus for SiP is the most pareto optimal point for energy-delay product (EDP) and cost. Neksus for SoC is slightly more energy-efficient (by 7.9%) due to SerDes removal but is significantly more expensive to develop. Overall, both Neksus SiP and Neksus SoC have lower EDP compared to regular SoC with memory reads/writes or SoC w/buf configuration, which bypasses memory and forwards data from caches/buffers at the memory controller.

6.6 Discussion

6.6.1 Scalability

Our proposed Neksus architecture assumes a centralized model with a single interconnect chiplet. However, since scalability is a limitation of crossbars solutions, we recommend a hierarchical approach comprising small interconnect chiplets distributed over the interposer. We envision multiple levels of interconnect chiplets, where those at the lowest level interface with IPs/mini-chains. These interconnect chiplets interface with other interconnect chiplets at higher levels of the hierarchy, adding latency of just one cycle for every hop between interconnect chiplet crossbars. IPs can be efficiently placed so that IPs communicating with each other are mapped to the same interconnect chiplet or logical siblings in the hierarchical tree. To improve the bandwidth of the system, either more micro-bumps or SerDes factor can be increased. SerDes accounts for only 2.7% of total power consumption, while DRAM is the major contributor. Neksus will reduce DRAM power consumption at a small overhead in network power due to additional SerDes power consumption. Another way to scale is to increase the size of the mini-chain. However, this will impact the delays in the system.

6.6.2 Chiplet Granularity

Instead of having separate chiplet for smaller functions/kernels, we can also aggregate multiple kernels to form one chiplet to reduce the total number of chiplet in the system. The decision regarding which chiplets should be combined depends on the communication pattern between chiplets for application traffic (i.e., chiplets frequently communicating should be combined), and chiplet size and number of chiplets in the system. Combining a lot of smaller chiplets can lead to a giant chiplet that will hit the system yield and defeats the whole purpose of disintegrating a monolithic chip for SiP architecture. Therefore, the granularity of the chiplet should be governed by the total yield of the system, which depends on the chiplet yield and stacking/bonding yield. Figure 6.12 shows the two different yields at different chip granularity and total system yield. For a

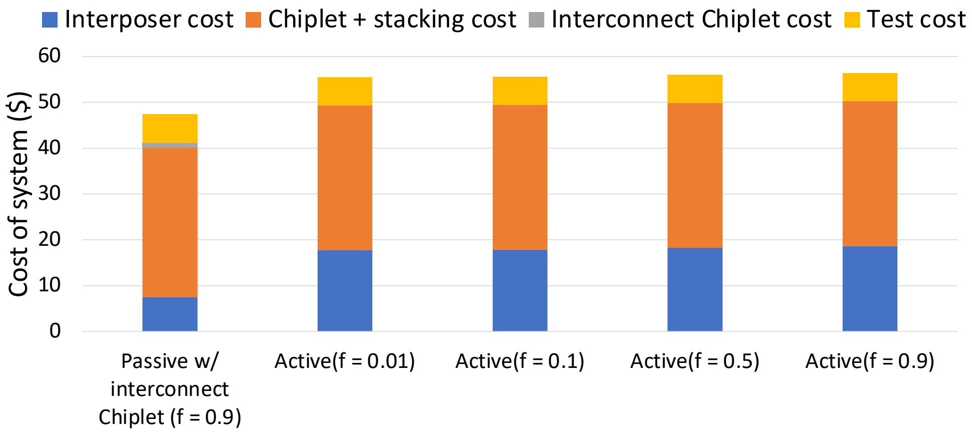


Figure 6.13: Breakdown of total system cost for passive and active interposer system.

256mm^2 monolithic system, as the size of the chiplet increases, the chiplet yield decreases but due to a decrease in the number of chiplets to stack/bond, the total bonding yield increases. Therefore, the total is maximum for 4mm^2 chiplet with 2mm^2 and 8mm^2 at almost par with the best. Neksus assumes a system of 64, 4mm^2 chiplet with 1, 8mm^2 interconnect IP, which is in sync with the chip granularity results from this sensitivity study.

6.6.3 Passive v/s Active Interposer

In this work, we focus on passive interposers because of their market feasibility, but the solution can be ported to active interposers as well. The benefits of the active interposer are because of buffering offered by the repeaters, due to which the maximum port bandwidth supported increases to 37 GBps (active interposer buffered every 1mm) from 32 GBps (passive interposer channel buffered at every 2mm at chiplet NI). Since performance is directly related to bandwidth, an active interposer can potentially improve the performance by 15% as compared to a passive interposer design. However, an active interposer is expensive and impacts the final retail cost of the system. As a counterargument, using separate chiplet in passive interposer also adds to the final cost of the system. Figure 6.13 shows the tradeoff of using active v/s passive interposer. To account for fabricating active transistors requiring more masks, we use an active interposer at 65nm with about 2.5× [207] cost as of passive interposer. Furthermore, for a fair comparison, instead of using a completely active layer of the interposer, we use a minimally active interposer [131] defined by the critical area (A_{cric}) in yield factor. As shown in Figure 6.13, using a cheaper passive interposer along with a small interconnect chiplet is economically better than a minimally active interposer with just 1% active area.

6.7 Conclusion

In this work, we explored the emerging System-in-Package (SiP) based designs to tackle critical challenges the industry is facing today. We identified a unique opportunity to design SiPs both to facilitate the modular plug-and-play integration of chiplets and to target the unique applications of heterogeneous mobile systems. We presented Neksus—a flexible interconnect architecture designed to mitigate the overheads of interposer-based 2.5D integration. Neksus included a dedicated interconnect chiplet that connects to IP chiplets via high-bandwidth mini-chains, which are particularly suitable for common data-flow patterns found in mobile applications. Our experimental evaluation showed that, for popular mobile SoC applications, Neksus provides up to 28% performance improvement and 31% energy savings over recent SiP architectures. Furthermore, we developed a detailed cost model that shows that SiPs reduce startup costs by approximately $5.2\times$ over System-on-Chip (SoC) designs while not increasing the recurring costs of mass manufacturing. Hence, in this chapter, we achieved this dissertation's goal of building performance and energy-efficient architecture for a low-cost SiP alternative to replace expensive SoCs, which are driving up the cost of edge platforms.

CHAPTER 7

Conclusion and Future Scope

In this chapter, we first provide a summary of the contributions proposed in this dissertation. We then discuss the potential future research directions for the work presented in this dissertation.

7.1 Summary of Contributions

Edge devices have grown rapidly in the past decade. They have evolved from lightweight sensing devices to portable mini-computers that we can wear or carry in our pockets. Simultaneously, edge applications are getting more intensive due to the rise of complex user-facing applications such as social media platforms, teleconferencing, fitness tracking, online shopping, and more, which have become a part of users' daily lives. However, despite advancements in the hardware design of edge devices, they are still unable to meet the increased computational demands of these heavy user applications. This dissertation aims to enhance the user experience on resource-constrained edge devices for such heavy user applications. To achieve this goal, we set out to improve the cost, performance, and energy efficiency of applications on such small resource-constrained edge devices.

The solutions developed in this dissertation first mitigate architectural bottlenecks of computing, memory, and network to improve the performance and energy of the existing devices. We then also offer performance and energy-efficient architecture for an upcoming low-cost hardware technology to reduce the cost of edge devices. We propose solutions that are guided by three principles: 1) cross-component optimizations across the system, 2) leverage user preference and characteristics in the hardware, and 3) co-design application and hardware for the edge system.

We first propose Seesaw, an energy-efficient end-to-end machine learning-based approach for IoT devices that leverages users' inherent sensing capabilities to trade-off applications' accuracy for lower compute and memory demand. This approach is based on the hypothesis that by looking at a visual output, like a video or a GPS route, a user cannot necessarily distinguish between the original and an intelligently modified, user-aware version of the visual output. We leverage the availability of multiple sensors on the edge device and offer an automated solution where low-power sensors, capturing user-activity, can govern the sensing rate of a high-power sensor without impacting the perceivable user experience. We employ a low-overhead decision tree predictor to learn such correlation automatically, and once the correlation is established, we predict the optimal sensing rate for the high-power sensor. This technique reduces the compute rate and memory storage requirements and therefore enhances the energy efficiency and battery life of the edge device. We show that we can improve the battery life of video recording devices by 32% and fitness trackers by 66% without significantly impacting the accuracy.

Our second solution, MyML, addresses the challenge of efficiently processing large and complex machine learning applications on the edge device. We observe that large models cater to a diverse set of users and are overkill to deploy on the user edge device since a single user does not have such variegated interests. In this work, we propose a hardware-friendly pruning mechanism to create small, user-specific ML models customized to users' preferences. The custom user-specific models have pruning granularity and precision governed by the underlying compute engine, such as a CPU or accelerator, which makes them highly performance- and energy-efficient. To the end, we also present an end-to-end collaborative system, which continuously learns the user preferences, builds the user-specific model, and updates the model when user preferences start to diverge. Our user-specific approach increases the average performance by 2.6× and reduces the average energy consumption by 1.9×.

Duet, the third work of this dissertation, focuses on accelerating the recommendation system that is the backbone of many popular user applications. A recommendation system ranks 100s of candidates to recommend only a few relevant items to the user, based on the user's preferences and properties. DNN-based recommendation models are extremely memory and compute-intensive, which makes it hard to adopt these models on edge devices. In this work, we decompose the monolithic recommendation model into user and item models, which are processed on the edge device and datacenter, respectively. The user model is computed only once for a user query, and its output (compute) is reused across all the candidates ranked by the item model to recommend only a few. Furthermore, we leverage user information that is readily available on the edge device to process the user model in a lightweight manner using scratchpad, memoization, and lower precision techniques. Therefore, the decomposition into a user model, its reuse, and lightweight edge computation helps us to enhance the performance and energy efficiency of the applications while also increasing privacy. We demonstrate that the proposed energy-efficient Duet architecture reduces the average latency by 6.4× and significantly improves energy efficiency by 4.6× across a wide range of recommendation models.

Finally, we propose Neksus, a low-cost design-time solution that improves performance and energy while also reducing the cost of edge devices. We present an interconnect architecture for SiP design that comprises cost-efficient chiplets stacked on a 2.5D interposer to overcome the cost of building a monolithic, expensive SoC. We demonstrate that the cost of building the SiP-based design is 5.2× lower than an SoC-based design. Moreover, our flexible interconnect architecture exposes the high-bandwidth interposer links to the hardware and efficiently maps bandwidth-intensive application data-flows over the new 2.5D architecture to extract performance and energy efficiency. Neksus increases the performance by 28% and improve energy efficiency by 31%.

To summarize, the proposed solutions in this dissertation enhance the user experience by: 1) Increasing performance or quality of service offered to the user, 2) Improving energy efficiency, which also lowers the total ownership cost, and 3) Building a flexible and efficient architecture for an emerging low-cost hardware alternative, which reduces the startup cost of developing edge devices.

7.2 Future Directions

In this section, we will discuss the future research directions in which we can expand the scope of this dissertation.

The methods established in this dissertation can be extended beyond the applications discussed in this thesis. Some of the emerging applications like Augmented Reality (AR) and Virtual Reality (VR) are envisioned to fit in a form factor of an eyeglass. However, the applications executed by an AR/VR device are extremely heavy. High-definition gaming/streaming, teleconferencing, localization, and hologram 3D rendering are a few examples of workloads of AR/VR applications. We believe that our solutions leveraging individual users' characteristics can be easily applied to these emerging applications. The fidelity of the applications will be highly dependent on the perceivable capabilities of the user; therefore, there is an opportunity to achieve better application efficiency by trading off accuracy to gain performance and energy, while still maintaining the fidelity of the application. Moreover, there is also an opportunity to leverage user information and propose user-aware hardware optimizations.

Another application that is gaining traction and proving its mettle in the automotive industry is the Advanced Driver Assisted System (ADAS). ADAS improves safety by actively providing information like incoming stop signs, lane detection, cruise control, collision detection, and more. ADAS is an ecosystem with multiple input sources like sensors (proximity, GPS, IMU, etc.), cameras, LiDARs, and, optionally, information received from nearby vehicles via vehicle-to-vehicle communication (V2X). ADAS-enabled vehicles also host a relatively powerful compute engine to process various tasks, which belong in the broad domain of computer vision, machine learning, point cloud analytics, etc. Such richness and diversity in the sensors offer a unique opportunity for cross-component optimizations across the system. There can be shared computation between different sensors or tasks working towards a common goal of mapping the nearby environment. The multiple input sources can communicate their findings to each other to improve performance, energy, and accuracy. There is also scope to balance the work distribution on edge and the datacenter for enhanced QoS.

Carbon emissions have become a major concern in the hardware community. There are ongoing efforts to study carbon footprints, and there is a push to build carbon-first systems [58, 94, 47]. In this dissertation, we have focused on the energy efficiency of edge devices; however, there is a scope to expand this body of work to focus on the carbon emissions of edge devices.

We see the potential in developing hybrid edge-cloud systems to reduce the total carbon footprint. Instead of optimizing the energy of applications, the hybrid system will optimize the carbon emissions by distributing the work intelligently on the device and a carbon-efficient datacenter. It will be the best of both worlds approach, where instead of processing entirely on either of the two computing ends (device or datacenter), which constantly erodes their life span, we can improve their combined life span and reduce the carbon footprint of renewing or replacing them. This approach also opens opportunities to judiciously select a carbon-efficient datacenter to complete the reduced amount of work, rather than choosing the fastest or the nearest datacenter to compute the entire application.

Furthermore, chiplet-based designs have immense potential to achieve the goal of sustainable hardware. Edge devices like smartphones are frequently discarded to upgrade their hardware, which offers better performance capabilities and battery life. This process creates massive e-waste problems and carbon emissions from frequent mass production. Chiplet-based architectures have the ability to upgrade each component individually instead of discarding the whole chip to change one piece, thus reducing e-waste and carbon emissions rapidly. We imagine chiplet technology to become the future for SoC, and thus there is a wide scope to study the emerging applications on these SiP platforms. It will bring with it new challenges & bottlenecks and, along with it, opportunities for new optimizations and innovative architecture.

BIBLIOGRAPHY

[1] Ad display/click data on taobao.com. https://tianchi.aliyun.com/dataset/dataDetail?dataId=56.

[2] Amazon astro, household robot for home monitoring, with alexa. https://www.theverge.com/23141966/amazon-astro-robot-review.

[3] Amazon echo show teardown. https://www.ifixit.com/Teardown/Amazon+Echo+Show+Teardown/94625.

[4] Apple watch series 6 teardown. https://www.ifixit.com/Teardown/Apple+Watch+Series+6+Teardown/136694.

[5] Arm cortex-m4 in a nutshell. https://www.st.com/content/st_com/en/arm-32-bit-microcontrollers/arm-cortex-m4.html.

[6] A battery-free sensor for underwater exploration. https://news.mit.edu/2019/battery-free-sensor-underwater-exploration-0820.

[7] Comparing the sizes of flagship chipsets: Qualcomm, samsung, huawei and apple. https://www.gsmarena.com/comparing_the_sizes_of_flagship_chipsets_qualcomm_samsung_huawei_adn_apple-news-30240.php.

[8] Criteo kaggele advertising dataset. https://ailab.criteo.com/ressources/.

[9] DARPA Program: Common Heterogeneous Integration and IP Reuse Strategies (CHIPS). https://www.darpa.mil/program/common-heterogeneous-integration-and-ip-reuse-strategies.

[10] Deep dive into netflix's recommender system. https://towardsdatascience.com/deep-dive-into-netflixs-recommender-system-341806ae3b48.

[11] Deep interest evolution network for click-through rate prediction framework. https://github.com/mouna99/dien.

[12] Edge tpu. https://cloud.google.com/edge-tpu.

[13] Edge tpu performance benchmarks. https://coral.ai/docs/edgetpu/benchmarks/.

[14] Google home teardown. https://www.ifixit.com/Teardown/Google+Home+Teardown/72684.

[15] How much will that chip cost? http://semiengineering.com/how-much-will-that-chip-cost/.

[16] How retailers can keep up with consumers. https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers.

[17] Inside amazon's ring alarm system. https://medium.com/tenable-techblog/inside-amazons-ring-alarm-system-9731bc519974.

[18] Intel image classification: Image scene classification of multiclass. https://www.kaggle.com/puneet6060/intel-image-classification/version/2.

[19] Internet of things (iot) and non-iot active device connections worldwide from 2010 to 2025. https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/.

[20] iphone 12 pro specifications. https://www.apple.com/iphone-12-pro/.

[21] iphone 14 pro specifications. https://www.apple.com/iphone-14-pro/specs/.

[22] Mobile apps usage reached an all-time high amidst stay-at-home measures due to covid-19 pandemic. https://www.appannie.com/en/insights/market-data/mobile-app-usage-surged-40-during-covid-19-pandemic/.

[23] Most popular apps (2022). https://www.businessofapps.com/data/most-popular-apps/.

[24] Number of smartphone users worldwide from 2016 to 2026. https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/.

[25] Oculus quest 2 now offers 120hz refresh rate and wireless pc streaming. https://www.tomsguide.com/news/oculus-quest-2-now-offers-120hz-refresh-rate-and-wireless-pc-streaming

[26] Samsung s22 specifications. https://www.samsung.com/global/galaxy/galaxy-s22/specs/.

[27] Snapdragon 855 mobile platform. https://www.qualcomm.com/products/application/smartphones/snapdragon-8-series-mobile-platforms/snapdragon-855-mobile-platform.

[28] Statista Reports. https://www.statista.com/statistics/1259878/edge-enabled-iot-device-market-worldwide/#:~:text=The%20number%20of%20consumer%20edge,%2C%20smartphones%2C%20to%20security%20cameras.

[29] Top apps of 2022 by installs, spend, and active users: Report. https://www.forbes.com/sites/johnkoetsier/2022/03/23/top-apps-of-2022-by-installs-spend-and-active-users-report/?sh=7cb533b4d3ac.

[30] User behavior data from taobao for recommendation. https://tianchi.aliyun.com/dataset/dataDetail?dataId=649.

[31] What is the npu in galaxy and what does it do? https://www.samsung.com/global/galaxy/what-is/npu/.

[32] Xnnpack. https://github.com/google/XNNPACK.

[33] Product specifications intel core2 duo processor e8400. https://ark.intel.com/products/33910/Intel-Core2-Duo-Processor-E8400-6M-Cache-3_00-GHz-1333-MHz-FSB, 2008.

[34] Axi, arm amba and axi, ace protocol specification. *AXI4, and AXI4-Lite, ACE and ACE-Lite, Technical report*, 2011.

[35] Chipworks provide first apple a7 die shot. https://www.anandtech.com/show/7355/chipworks-provides-first-apple-a7-die-shot, 2013.

[36] Frdm-k64f. https://os.mbed.com/platforms/FRDM-K64F/, 2014.

[37] AMD Radeion R9 Fury X Review. http://www.anandtech.com/show/9390/the-amd-radeon-r9-fury-x-review/3, 2015.

[38] Nvidia Tegra X1 white paper. http://international.download.nvidia.com/pdf/tegra/Tegra-X1-whitepaper-v1.0.pdf, 2015.

[39] GoPro reveals what the GPS in the Hero5 Black camera is for. https://www.cnet.com/news/gopro-unlocks-hero5-black-telemetry-data-for-video-overlays/, 2016.

[40] VMAF - Video Multi-Method Assessment Fusion. https://github.com/Netflix/vmaf, 2016.

[41] Gps watch data. https://www.kaggle.com/jsphyg/weather-dataset-rattle-package, 2017.

[42] Arable Agricultural IoT platform. https://www.arable.com/, 2018.

[43] Fitbit battery life. https://help.fitbit.com/articles/en_US/Help_article/2004, 2018.

[44] FitBit Products. https://www.fitbit.com/us/shop/surge, 2018.

[45] Rain in australia. https://www.kaggle.com/antgoldbloom/gps-watch-data, 2018.

[46] Smart parking product brief. http://www.libelium.com/products/smart-parking/, 2018.

[47] Bilge Acun, Benjamin Lee, Kiwan Maeng, Manoj Chakkaravarthy, Udit Gupta, David Brooks, and Carole-Jean Wu. A holistic approach for designing carbon aware datacenters. *arXiv preprint arXiv:2201.10036*, 2022.

[48] Bilge Acun, Matthew Murphy, Xiaodong Wang, Jade Nie, Carole-Jean Wu, and Kim Hazelwood. Understanding training efficiency of deep learning recommendation models at scale. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 802–814. IEEE, 2021.

[49] Muhammad Adnan, Yassaman Ebrahimzadeh Maboud, Divya Mahajan, and Prashant J. Nair. Accelerating recommendation system training by leveraging popular choices. 15(1):127–140, sep 2021.

[50] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Cnvlutin: Ineffectual-neuron-free deep neural network computing. *ACM SIGARCH Computer Architecture News*, 44(3):1–13, 2016.

[51] Pavlos Athanasios Apostolopoulos, Eirini Eleni Tsiropoulou, and Symeon Papavassiliou. Risk-aware data offloading in multi-server multi-access edge computing environment. *IEEE/ACM Transactions on Networking*, 28(3):1405–1418, 2020.

[52] Bahar Asgari, Ramyad Hadidi, Jiashen Cao, Sung-Kyu Lim, Hyesoon Kim, et al. Fafnir: Accelerating sparse gathering by using efficient near-memory intelligent reduction. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 908–920. IEEE, 2021.

[53] Anubhav Ashok, Nicholas Rhinehart, Fares Beainy, and Kris M Kitani. N2n learning: Network to network compression via policy gradient reinforcement learning. *arXiv preprint arXiv:1709.06030*, 2017.

[54] Todd Austin. Preparing for a post moore's law world. *Keynote at International Symposium of Microarchitecture (MICRO-48)*, 2015. https://www.microarch.org/micro48/files/slides/Keynote-II.pdf.

[55] Suat U Ay. A 1.32 pw/frame•pixel 1.2 v cmos energy-harvesting and imaging (ehi) aps imager. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*, pages 116–118. IEEE, 2011.

[56] Carmen Badea, Mohammad R Haghighat, Alexandru Nicolau, and Alexander V Veidenbaum. Towards parallelizing the layout engine of firefox. *Proc. of USENIX HotPar*, 2010.

[57] James Balfour and William J Dally. Design tradeoffs for tiled cmp on-chip networks. In *ACM International Conference on Supercomputing (ICS)*, pages 187–198. ACM, 2006.

[58] Noman Bashir, Tian Guo, Mohammad Hajiesmaili, David Irwin, Prashant Shenoy, Ramesh Sitaraman, Abel Souza, and Adam Wierman. Enabling sustainable clouds: The case for virtualizing the energy system. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 350–358, 2021.

[59] Vikram Bhatt, Nathan Goulding-Hotta, Qiaoshi Zheng, Jack Sampson, Steven Swanson, and Michael Bedford Taylor. Sichrome: Mobile web browsing in hardware to save energy. In *DaSi: First Dark Silicon Workshop*, 2012.

[60] Keith Bonawitz et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

[61] Mark Buckler, Philip Bedoukian, Suren Jayasuriya, and Adrian Sampson. Eva^2: Exploiting temporal redundancy in live computer vision. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 533–546. IEEE, 2018.

[62] Michael Butkiewicz, Daimeng Wang, Zhe Wu, Harsha V Madhyastha, and Vyas Sekar. Klotski: Reprioritizing web content to improve user experience on mobile devices. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 439–453, 2015.

[63] Han Cai, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. Path-level network transformation for efficient architecture search. In *International Conference on Machine Learning*, pages 678–687. PMLR, 2018.

[64] Luiz Ceze, Mark Hill, Karu Sankaralingam, and Thomas Wenisch. Democratizing design for future computing platforms. *CRA Whitepapers*, 2017. http://cra.org/ccc/resources/ccc-led-whitepapers/.

[65] Niladrish Chatterjee, Mike O'Connor, Donghyuk Lee, Daniel R Johnson, Stephen W Keckler, Minsoo Rhu, and William J Dally. Architecting an energy-efficient dram system for gpus. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 73–84. IEEE, 2017.

[66] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.

[67] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM SIGARCH Computer Architecture News*, 42(1):269–284, 2014.

[68] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 367–379, 2016.

[69] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits*, 52(1):127–138, 2016.

[70] Hari Cherupalli, Henry Duwe, Weidong Ye, Rakesh Kumar, and John Sartori. Bespoke processors for applications with ultra-low area and power constraints. In *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*, pages 41–54. IEEE, 2017.

[71] Hari Cherupalli, Henry Duwe, Weidong Ye, Rakesh Kumar, and John Sartori. Determining application-specific peak power and energy requirements for ultra-low-power processors. *ACM Transactions on Computer Systems (TOCS)*, 35(3):9, 2017.

[72] Nachiappan Chidambaram Nachiappan, Praveen Yedlapalli, Niranjan Soundararajan, Mahmut Taylan Kandemir, Anand Sivasubramaniam, and Chita R Das. Gemdroid: A framework to evaluate mobile platforms. *ACM SIGMETRICS Performance Evaluation Review*, 42(1):355–366, 2014.

[73] James A Cunningham. The use and evaluation of yield models in integrated circuit manufacturing. *IEEE Transactions on Semiconductor Manufacturing*, 3(2):60–71, 1990.

[74] Alberto Delmas Lascorz, Patrick Judd, Dylan Malone Stuart, Zissis Poulos, Mostafa Mahmoud, Sayeh Sharify, Milos Nikolic, Kevin Siu, and Andreas Moshovos. Bit-tactical: A software/hardware approach to exploiting value and bit sparsity in neural networks. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 749–763. ACM, 2019.

[75] Chunhua Deng, Siyu Liao, Yi Xie, Keshab K Parhi, Xuehai Qian, and Bo Yuan. Permdnn: Efficient compressed dnn architecture with permuted diagonal matrices. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 189–202. IEEE, 2018.

[76] Chunhua Deng, Fangxuan Sun, Xuehai Qian, Jun Lin, Zhongfeng Wang, and Bo Yuan. Tie: energy-efficient tensor train-based inference engine for deep neural network. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 264–278. ACM, 2019.

[77] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[78] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. Shidiannao: Shifting vision processing closer to the sensor. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 92–104. ACM, 2015.

[79] Masoumeh Ebrahimi, Awet Yemane Weldezion, and Masoud Daneshtalab. Nod: Network-on-die as a standalone noc for heterogeneous many-core systems in 2.5 d ics. In *2017 19th International Symposium on Computer Architecture and Digital Systems (CADS)*, pages 1–6. IEEE, 2017.

[80] Pete Ehrett, Todd Austin, and Valeria Bertacco. Sipterposer: A fault-tolerant substrate for flexible system-in-package design. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 510–515. IEEE, 2019.

[81] Assaf Eisenman, Maxim Naumov, Darryl Gardner, Misha Smelyanskiy, Sergey Pupyrev, Kim Hazelwood, Asaf Cidon, and Sachin Katti. Bandana: Using non-volatile memory for storing deep learning models. *Proceedings of Machine Learning and Systems*, 1:40–52, 2019.

[82] David Fleet and Yair Weiss. Optical flow estimation. In *Handbook of mathematical models in computer vision*, pages 237–257. Springer, 2006.

[83] Harry D Foster. Trends in functional verification: a 2014 industry study. In *Proceedings of the 52nd Annual Design Automation Conference*, page 48. ACM, 2015.

[84] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, et al. A configurable cloud-scale dnn processor for real-time ai. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–14. IEEE, 2018.

[85] Yu Gong, Ziwen Jiang, Yufei Feng, Binbin Hu, Kaiqi Zhao, Qingwen Liu, and Wenwu Ou. Edgerec: recommender system on edge in mobile taobao. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 2477–2484, 2020.

[86] Zhangxiaowen Gong, Houxiang Ji, Christopher W Fletcher, Christopher J Hughes, Sara Baghsorkhi, and Josep Torrellas. Save: Sparsity-aware vector engine for accelerating dnn training and inference on cpus. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 796–810. IEEE, 2020.

[87] Zhangxiaowen Gong, Houxiang Ji, Christopher W Fletcher, Christopher J Hughes, and Josep Torrellas. Sparsetrain: Leveraging dynamic sparsity in software for training dnns on general-purpose simd processors. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, pages 279–292, 2020.

[88] Vidushi Goyal, Valeria Bertacco, and Reetuparna Das. Seesaw: End-to-end dynamic sensing for iot using machine learning. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–19. IEEE, 2020.

[89] Vidushi Goyal, Valeria Bertacco, and Reetuparna Das. Myml: User-driven machine learning. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021.

[90] Vidushi Goyal, Reetuparna Das, and Valeria Bertacco. Hardware-friendly user-specific machine learning for edge devices. *ACM Transactions on Embedded Computing Systems (TECS)*, 2022.

[91] Vidushi Goyal, Xiaowei Wang, Valeria Bertacco, and Reetuparna Das. Neksus: An interconnect for heterogeneous system-in-package architectures. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 12–21. IEEE, 2020.

[92] Laura M Grupp, Adrian M Caulfield, Joel Coburn, Steven Swanson, Eitan Yaakobi, Paul H Siegel, and Jack K Wolf. Characterizing flash memory: Anomalies, observations, and applications. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 24–33, 2009.

[93] Zheng Guo, Daeyeon Kim, Satyanand Nalam, Jami Wiedemer, Xiaofei Wang, and Eric Karl. A 23.6-mb/mm^{2} sram in 10-nm finfet technology with pulsed-pmos tvc and stepped-wl for low-voltage applications. *IEEE Journal of Solid-State Circuits*, 54(1):210–216, 2018.

[94] Udit Gupta, Mariam Elgamal, Gage Hills, Gu-Yeon Wei, Hsien-Hsin S Lee, David Brooks, and Carole-Jean Wu. Act: designing sustainable computer systems with an architectural carbon modeling tool. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 784–799, 2022.

[95] Udit Gupta, Samuel Hsia, Vikram Saraph, Xiaodong Wang, Brandon Reagen, Gu-Yeon Wei, Hsien-Hsin S Lee, David Brooks, and Carole-Jean Wu. Deeprecsys: A system for optimizing end-to-end at-scale neural recommendation inference. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 982–995. IEEE, 2020.

[96] Udit Gupta, Samuel Hsia, Jeff Zhang, Mark Wilkening, Javin Pombra, Hsien-Hsin Sean Lee, Gu-Yeon Wei, Carole-Jean Wu, and David Brooks. Recpipe: Co-designing models and hardware to jointly optimize recommendation quality and performance. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 870–884, 2021.

[97] Udit Gupta, Carole-Jean Wu, Xiaodong Wang, Maxim Naumov, Brandon Reagen, David Brooks, Bradford Cottel, Kim Hazelwood, Mark Hempstead, Bill Jia, et al. The architectural implications of facebook's dnn-based personalized recommendation. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 488–501. IEEE, 2020.

[98] Ramyad Hadidi, Jiashen Cao, Yilun Xie, Bahar Asgari, Tushar Krishna, and Hyesoon Kim. Characterizing the deployment of deep neural networks on commercial edge devices. In *2019 IEEE International Symposium on Workload Characterization (IISWC)*, pages 35–48. IEEE, 2019.

[99] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, et al. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 75–84, 2017.

[100] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News*, 44(3):243–254, 2016.

[101] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[102] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 1*, pages 1135–1143, 2015.

[103] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, et al. Applied machine learning at facebook: A datacenter infrastructure perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 620–629. IEEE, 2018.

[104] Hengtao He, Chao-Kai Wen, Shi Jin, and Geoffrey Ye Li. Model-driven deep learning for mimo detection. *IEEE Transactions on Signal Processing*, 68:1702–1715, 2020.

[105] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[106] Songtao He, Yunxin Liu, and Hucheng Zhou. Optimizing smartphone power consumption through dynamic resolution scaling. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pages 27–39. ACM, 2015.

[107] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European conference on computer vision (ECCV)*, pages 784–800, 2018.

[108] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017.

[109] Kartik Hegde, Jiyong Yu, Rohit Agrawal, Mengjia Yan, Michael Pellauer, and Christopher W Fletcher. Ucnn: Exploiting computational reuse in deep neural networks via weight repetition. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, pages 674–687. IEEE Press, 2018.

[110] Mark Hill and Vijay Janapa Reddi. Gables: A roofline model for mobile socs. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 317–330. IEEE, 2019.

[111] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[112] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[113] Connor Holmes, Daniel Mawhirter, Yuxiong He, Feng Yan, and Bo Wu. Grnn: Low-latency and scalable rnn inference on gpus. In *Proceedings of the Fourteenth EuroSys Conference 2019*, pages 1–16, 2019.

[114] Ang-Chih Hsieh and TingTing Hwang. Tsv redundancy: Architecture and design issues in 3-d ic. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(4):711–722, 2012.

[115] Chao-Tsung Huang, Yu-Chun Ding, Huan-Ching Wang, Chi-Wen Weng, Kai-Ping Lin, Li-Wei Wang, and Li-De Chen. ecnn: A block-based and highly-parallel cnn accelerator for edge inference. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 182–195, 2019.

[116] Xin-Lin Huang, Xiaomin Ma, and Fei Hu. Machine learning and intelligent communications. *Mobile Networks and Applications*, 23(1):68–70, 2018.

[117] Muhammad Huzaifa, Rishi Desai, Samuel Grayson, Xutao Jiang, Ying Jing, Jae Lee, Fang Lu, Yihan Pang, Joseph Ravichandran, Finn Sinclair, et al. Exploring extended reality with illixr: A new playground for architecture research. *arXiv preprint arXiv:2004.04643*, 2020.

[118] Ranggi Hwang, Taehun Kim, Youngeun Kwon, and Minsoo Rhu. Centaur: A chiplet-based, hybrid sparse-dense accelerator for personalized recommendations. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 968–981. IEEE, 2020.

[119] Animesh Jain, Amar Phanishayee, Jason Mars, Lingjia Tang, and Gennady Pekhimenko. Gist: Efficient data encoding for deep neural network training. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 776–789. IEEE, 2018.

[120] Natalie Enright Jerger, Ajaykumar Kannan, Zimo Li, and Gabriel H Loh. Noc architectures for silicon interposer systems: Why pay for more wires when you can get them (from your interposer) for free? In *47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 458–470. IEEE, 2014.

[121] Djordje Jevdjic, Karin Strauss, Luis Ceze, and Henrique S Malvar. Approximate storage of compressed and encrypted videos. *ACM SIGOPS Operating Systems Review*, 51(2):361–373, 2017.

[122] Bo Jiang and Ying Sha. Modeling temporal dynamics of user interests in online social networks. *Procedia Computer Science*, 51:503–512, 2015.

[123] Nan Jiang, Daniel U Becker, George Michelogiannakis, James Balfour, Brian Towles, David E Shaw, John Kim, and William J Dally. A detailed and flexible cycle-accurate network-on-chip simulator. In *2013 IEEE international symposium on performance analysis of systems and software (ISPASS)*, pages 86–96. IEEE, 2013.

[124] Wenqi Jiang, Zhenhao He, Shuai Zhang, Thomas B Preußer, Kai Zeng, Liang Feng, Jiansong Zhang, Tongxuan Liu, Yong Li, Jingren Zhou, et al. Microrec: efficient recommendation inference by hardware and data structure solutions. *Proceedings of Machine Learning and Systems*, 3:845–859, 2021.

[125] Antonio R Jimenez, Fernando Seco, Carlos Prieto, and Jorge Guevara. A comparison of pedestrian dead-reckoning algorithms using a low-cost mems imu. In *Intelligent Signal Processing, 2009. WISP 2009. IEEE International Symposium on*, pages 37–42. IEEE, 2009.

[126] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

[127] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh, and Daniel Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. In *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 96–107, 2002.

[128] Hongju Kal, Seokmin Lee, Gun Ko, and Won Woo Ro. Space: locality-aware processing in heterogeneous memory for personalized recommendations. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 679–691. IEEE, 2021.

[129] Uksong Kang, Hoe-Ju Chung, Seongmoo Heo, Duk-Ha Park, Hoon Lee, Jin Ho Kim, Soon-Hong Ahn, Soo-Ho Cha, Jaesung Ahn, DukMin Kwon, et al. 8 gb 3-d ddr3 dram using through-silicon-via technology. *IEEE Journal of Solid-State Circuits*, 45(1):111–119, 2010.

[130] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGPLAN Notices*, 52(4):615–629, 2017.

[131] Ajaykumar Kannan, Natalie Enright Jerger, and Gabriel H Loh. Enabling interposer-based disintegration of multi-core processors. In *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 546–558. IEEE, 2015.

[132] M Ataul Karim, Paul D Franzon, and Anil Kumar. Power comparison of 2d, 3d and 2.5 d interconnect solutions and power optimization of interposer interconnects. In *2013 IEEE 63rd Electronic Components and Technology Conference*, pages 860–866. IEEE, 2013.

[133] Liu Ke, Udit Gupta, Benjamin Youngjae Cho, David Brooks, Vikas Chandra, Utku Diril, Amin Firoozshahian, Kim Hazelwood, Bill Jia, Hsien-Hsin S Lee, et al. Recnmp: Accelerating personalized recommendation with near-memory processing. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 790–803. IEEE, 2020.

[134] Conor Kelton, Jihoon Ryoo, Aruna Balasubramanian, and Samir R Das. Improving user perceived page load times using gaze. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 545–559, 2017.

[135] Mehrdad Khani, Mohammad Alizadeh, Jakob Hoydis, and Phil Fleming. Adaptive neural signal detection for massive mimo. *IEEE Transactions on Wireless Communications*, 19(8):5635–5648, 2020.

[136] Moein Khazraee, Lu Zhang, Luis Vega, and Michael Bedford Taylor. Moonwalk: Nre optimization in asic clouds. *ACM SIGARCH Computer Architecture News*, 45(1):511–526, 2017.

[137] Donnie H Kim, Younghun Kim, Deborah Estrin, and Mani B Srivastava. Sensloc: sensing everyday places and paths using less energy. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, pages 43–56, 2010.

[138] John Kim, James Balfour, and William Dally. Flattened butterfly topology for on-chip networks. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 172–182. IEEE Computer Society, 2007.

[139] Youngwoo Kim, Jonghyun Cho, Kiyeong Kim, Venky Sundaram, Rao Tummala, and Joungho Kim. Signal and power integrity analysis in 2.5 d integrated circuits (ics) with glass, silicon and organic interposer. In *2015 IEEE 65th Electronic Components and Technology Conference (ECTC)*, pages 738–743. IEEE, 2015.

[140] Mikkel Baun Kjærgaard, Sourav Bhattacharya, Henrik Blunck, and Petteri Nurmi. Energy-efficient trajectory tracking for mobile devices. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 307–320. ACM, 2011.

[141] Mikkel Baun Kjærgaard, Jakob Langdal, Torben Godsk, and Thomas Toftkjær. Entracked: energy-efficient robust position tracking for mobile devices. In *Proceedings of the 7th international conference on Mobile systems, applications, and services*, pages 221–234. ACM, 2009.

[142] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[143] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[144] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[145] Gokul Kumar, Tapobrata Bandyopadhyay, Vijay Sukumaran, Venky Sundaram, Sung Kyu Lim, and Rao Tummala. Ultra-high i/o density glass/silicon interposers for high bandwidth smart mobile applications. In *2011 IEEE 61st Electronic Components and Technology Conference (ECTC)*, pages 217–223. IEEE, 2011.

[146] Youngeun Kwon, Yunjae Lee, and Minsoo Rhu. Tensordimm: A practical near-memory processing architecture for embeddings and tensor operations in deep learning. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 740–753, 2019.

[147] Youngeun Kwon, Yunjae Lee, and Minsoo Rhu. Tensor casting: Co-designing algorithm-architecture for personalized recommendation training. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 235–248. IEEE, 2021.

[148] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[149] Chang-Chi Lee, CP Hung, Calvin Cheung, Ping-Feng Yang, Chin-Li Kao, Dao-Long Chen, Meng-Kai Shih, Chien-Lin Chang Chien, Yu-Hsiang Hsiao, Li-Chieh Chen, et al. An overview of the development of a gpu with integrated hbm on silicon interposer. In *2016 IEEE 66th Electronic Components and Technology Conference (ECTC)*, pages 1439–1444. IEEE, 2016.

[150] Yejin Lee, Seong Hoon Seo, Hyunji Choi, Hyoung Uk Sul, Soosung Kim, Jae W Lee, and Tae Jun Ham. Merci: efficient embedding reduction on commodity hardware via sub-query memoization. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 302–313, 2021.

[151] Robert W Levi and Thomas Judd. Dead reckoning navigational system using accelerometer to measure foot impacts, December 10 1996. US Patent 5,583,776.

[152] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[153] HY Li, Guruprasad Katti, L Ding, Surya Bhattacharya, and GQ Lo. The cost study of 300mm through silicon interposer (tsi) with beol interconnect. In *2013 IEEE 15th Electronics Packaging Technology Conference (EPTC 2013)*, pages 664–668. IEEE, 2013.

[154] Daniyal Liaqat, Silviu Jingoi, Eyal de Lara, Ashvin Goel, Wilson To, Kevin Lee, Italo De Moraes Garcia, and Manuel Saldana. Sidewinder: An energy efficient and developer friendly heterogeneous architecture for continuous mobile sensing. volume 44, pages 205–215. ACM, 2016.

[155] Robert LiKamWa, Yunhui Hou, Julian Gao, Mia Polansky, and Lin Zhong. Redeye: analog convnet image sensor architecture for continuous mobile vision. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 255–266. IEEE Press, 2016.

[156] Robert LiKamWa, Bodhi Priyantha, Matthai Philipose, Lin Zhong, and Paramvir Bahl. Energy characterization and optimization of image sensing toward continuous mobile vision. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 69–82. ACM, 2013.

[157] Kaisen Lin, Aman Kansal, Dimitrios Lymberopoulos, and Feng Zhao. Energy-accuracy trade-off for continuous mobile device location. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 285–298. ACM, 2010.

[158] Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md E Haque, Lingjia Tang, and Jason Mars. The architectural implications of autonomous driving: Constraints and acceleration. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 751–766, 2018.

[159] Jie Liu, Bodhi Priyantha, Ted Hart, Heitor S Ramos, Antonio AF Loureiro, and Qiang Wang. Energy efficient gps sensing with cloud offloading. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, pages 85–98. ACM, 2012.

[160] Igor Loi, Subhasish Mitra, Thomas H Lee, Shinobu Fujita, and Luca Benini. A low-overhead fault tolerance scheme for tsv-based 3d network on chip links. In *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 598–602. IEEE Press, 2008.

[161] Changqing Luo, Jinlong Ji, Qianlong Wang, Xuhui Chen, and Pan Li. Channel state information prediction for 5g wireless communications: A deep learning approach. *IEEE Transactions on Network Science and Engineering*, 7(1):227–236, 2018.

[162] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[163] Sangkug Lym, Esha Choukse, Siavash Zangeneh, Wei Wen, Sujay Sanghavi, and Mattan Erez. Prunetrain: fast neural network training by dynamic sparse model reconfiguration. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13, 2019.

[164] Mostafa Mahmoud, Isak Edo, Ali Hadi Zadeh, Omar Mohamed Awad, Gennady Pekhimenko, Jorge Albericio, and Andreas Moshovos. Tensordash: Exploiting sparsity to accelerate deep neural network training. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 781–795. IEEE, 2020.

[165] Vikram Sharma Mailthody, Zaid Qureshi, Weixin Liang, Ziyan Feng, Simon Garcia De Gonzalo, Youjie Li, Hubertus Franke, Jinjun Xiong, Jian Huang, and Wen-mei Hwu. Deepstore: In-storage acceleration for intelligent queries. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 224–238, 2019.

[166] Peter Mattson, Vijay Janapa Reddi, Christine Cheng, Cody Coleman, Greg Diamos, David Kanter, Paulius Micikevicius, David Patterson, Guenther Schmuelling, Hanlin Tang, et al. Mlperf: An industry standard benchmark suite for machine learning performance. *IEEE Micro*, 40(2):8–16, 2020.

[167] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th international*

ACM SIGIR conference on research and development in information retrieval, pages 43–52, 2015.

[168] David Metcalf, Sharlin TJ Milliard, Melinda Gomez, and Michael Schwartz. Wearables and the internet of things for health: Wearable, interconnected devices promise more efficient and comprehensive health care. volume 7, pages 35–39. IEEE, 2016.

[169] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi. Cacti 6.0: A tool to model large caches. *HP Laboratories*, pages 22–31, 2009.

[170] Nachiappan Chidambaram Nachiappan, Haibo Zhang, Jihyun Ryoo, Niranjan Soundararajan, Anand Sivasubramaniam, Mahmut T Kandemir, Ravi Iyer, and Chita R Das. Vip: virtualizing ip chains on handheld platforms. *ACM SIGARCH Computer Architecture News*, 43(3):655–667, 2016.

[171] Saman Naderiparizi, Pengyu Zhang, Matthai Philipose, Bodhi Priyantha, Jie Liu, and Deepak Ganesan. Glimpse: A programmable early-discard camera architecture for continuous mobile vision. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, pages 292–305. ACM, 2017.

[172] Omar Naji, Christian Weis, Matthias Jung, Norbert Wehn, and Andreas Hansson. A high-level dram timing, power and area exploration tool. In *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, pages 149–156. IEEE, 2015.

[173] Arvind Narayanan, Xumiao Zhang, Ruiyang Zhu, Ahmad Hassan, Shuowei Jin, Xiao Zhu, Xiaoxuan Zhang, Denis Rybkin, Zhengxuan Yang, Zhuoqing Morley Mao, et al. A variegated look at 5g in the wild: performance, power, and qoe implications. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 610–625, 2021.

[174] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091*, 2019.

[175] NCSU. Freepdk 45nm. `Available:https://www.eda.ncsu.edu/wiki/FreePDK`.

[176] Ravi Netravali, Ameesh Goyal, James Mickens, and Hari Balakrishnan. Polaris: Faster page loads using fine-grained dependency tracking. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, 2016.

[177] Wei Niu, Xiaolong Ma, Sheng Lin, Shihao Wang, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 907–922, 2020.

[178] Kent W Nixon, Xiang Chen, and Yiran Chen. Scope-quality retaining display rendering workload scaling based on user-smartphone distance. In *Computer-Aided Design (ICCAD), 2016 IEEE/ACM International Conference on*, pages 1–6. IEEE, 2016.

[179] Jeongyeup Paek, Joongheon Kim, and Ramesh Govindan. Energy-efficient rate-adaptive gps-based positioning for smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 299–314. ACM, 2010.

[180] Saptadeep Pal, Daniel Petrisko, Adeel A Bajwa, Puneet Gupta, Subramanian S Iyer, and Rakesh Kumar. A case for packageless processors. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 466–479. IEEE, 2018.

[181] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. Scnn: An accelerator for compressed-sparse convolutional neural networks. In *Proceedings of 44th International Symposium on Computer Architecture*, pages 27–40. ACM, 2017.

[182] Eunhyeok Park, Dongyoung Kim, and Sungjoo Yoo. Energy-efficient neural network accelerator based on outlier-aware low-precision computation. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 688–698. IEEE, 2018.

[183] Hyo Park, Yunah Shin, Se Choi, and Yousok Kim. An integrative structural health monitoring system for the local/global responses of a large-scale irregular building under construction. volume 13, pages 9085–9103. Multidisciplinary Digital Publishing Institute, 2013.

[184] Jaehyun Park, Byeongho Kim, Sungmin Yun, Eojin Lee, Minsoo Rhu, and Jung Ho Ahn. Trim: Enhancing processor-memory interfaces with scalable tensor reduction in memory. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 268–281, 2021.

[185] John W Poulton, William J Dally, Xi Chen, John G Eyles, Thomas H Greer, Stephen G Tell, John M Wilson, and C Thomas Gray. A 0.54 pj/b 20 gb/s ground-referenced single-ended short-reach serial link in 28 nm cmos for advanced packaging applications. *IEEE Journal of Solid-State Circuits*, 48(12):3206–3218, 2013.

[186] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 267–278. IEEE, 2016.

[187] Vijay Janapa Reddi, Hongil Yoon, and Allan Knies. Two billion devices and counting. *IEEE Micro*, 38(1):6–21, 2018.

[188] Marc Riera, Jose-Maria Arnau, and Antonio González. Computation reuse in dnns by exploiting input similarity. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 57–68. IEEE, 2018.

[189] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[190] Kirk Saban. Xilinx stacked silicon interconnect technology delivers breakthrough fpga capacity, bandwidth, and power efficiency. *Whitepaper*, 2012.

[191] Hasim Sak, Andrew W Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. 2014.

[192] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. Scale-sim: Systolic cnn accelerator simulator. *arXiv preprint arXiv:1811.02883*, 2018.

[193] Mohammad Samragh, Mohammad Ghasemzadeh, and Farinaz Koushanfar. Customizing neural networks for efficient fpga implementation. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 85–92. IEEE, 2017.

[194] Neev Samuel, Tzvi Diskin, and Ami Wiesel. Deep mimo detection. In *2017 IEEE 18th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 1–5. IEEE, 2017.

[195] Sudhir Satpathy, Korey Sewell, Thomas Manville, Yen-Po Chen, Ronald G. Dreslinski, Dennis Sylvester, Trevor N. Mudge, and David Blaauw. A 4.5tb/s 3.4tb/s/w 64x64 switch fabric with self-updating least recently granted priority and quality of service arbitration in 45nm cmos. In *ISSCC*, 2012.

[196] Geet Sethi, Bilge Acun, Niket Agarwal, Christos Kozyrakis, Caroline Trippel, and Carole-Jean Wu. Recshard: Statistical feature-based memory optimization for industry-scale neural recommendation. *arXiv preprint arXiv:2201.10095*, 2022.

[197] Korey Sewell, Ronald G. Dreslinski, Thomas Manville, Sudhir Satpathy, Nathaniel Ross Pinckney, Geoffrey Blake, Michael Cieslak, Reetuparna Das, Thomas F. Wenisch, Dennis Sylvester, David Blaauw, and Trevor N. Mudge. Swizzle-switch networks for many-core systems. *IEEE J. Emerg. Sel. Topics Circuits Syst.*, 2012.

[198] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News*, 44(3):14–26, 2016.

[199] Yakun Sophia Shao, Brandon Reagen, Gu-Yeon Wei, and David Brooks. The aladdin approach to accelerator design and modeling. *IEEE Micro*, 35(3):58–70, 2015.

[200] Sayeh Sharify, Alberto Delmas Lascorz, Mostafa Mahmoud, Milos Nikolic, Kevin Siu, Dylan Malone Stuart, Zissis Poulos, and Andreas Moshovos. Laconic deep learning inference acceleration. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 304–317. ACM, 2019.

[201] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Vikas Chandra, and Hadi Esmaeilzadeh. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural networks. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, pages 764–775. IEEE Press, 2018.

[202] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. *Advances in neural information processing systems*, 27, 2014.

[203] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[204] Mingcong Song, Kan Zhong, Jiaqi Zhang, Yang Hu, Duo Liu, Weigong Zhang, Jing Wang, and Tao Li. In-situ ai: Towards autonomous and incremental deep learning for iot systems. In *2018 IEEE InternatiOnal SympOsium On High PerfOrmance COmputer Architecture (HPCA)*, pages 92–103. IEEE, 2018.

[205] J.E.D.E.C. Standard. Low power double data rate 4. *LPDDR4*, 2014.

[206] Dylan Stow, Itir Akgun, Russell Barnes, Peng Gu, and Yuan Xie. Cost analysis and cost-driven ip reuse methodology for soc design based on 2.5 d/3d integration. In *Proceedings of the 35th International Conference on Computer-Aided Design*, page 56. ACM, 2016.

[207] Dylan Stow, Yuan Xie, Taniya Siddiqua, and Gabriel H Loh. Cost-effective design of scalable high-performance systems using active and passive interposers. In *Proceedings of the 36th International Conference on Computer-Aided Design*, pages 728–735. IEEE Press, 2017.

[208] Vijay Sukumaran, Tapobrata Bandyopadhyay, Venky Sundaram, and Rao Tummala. Low-cost thin glass interposers as a superior alternative to silicon and organic interposers for packaging of 3-d ics. *IEEE Transactions on Components, Packaging and Manufacturing Technology*, 2(9):1426–1433, 2012.

[209] Vivienne Sze et al. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[210] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[211] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[212] Mottaqiallah Taouil, Said Hamdioui, Erik Jan Marinissen, and Sudipta Bhawmik. Using 3d-costar for 2.5 d test cost optimization. In *IEEE International 3D Systems Integration Conference (3DIC)*, pages 1–8. IEEE, 2013.

[213] Aosen Wang, Lizhong Chen, and Wenyao Xu. Xpro: A cross-end processing architecture for data analytics in wearables. In *ACM SIGARCH Computer Architecture News*, volume 45, pages 69–80. ACM, 2017.

[214] Siqi Wang, Gayathri Ananthanarayanan, Yifan Zeng, Neeraj Goel, Anuj Pathania, and Tulika Mitra. High-throughput cnn inference on embedded arm big. little multicore processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2254–2267, 2019.

[215] Xiaowei Wang, Vidushi Goyal, Jiecao Yu, Valeria Bertacco, Andrew Boutros, Eriko Nurvitadhi, Charles Augustine, Ravi Iyer, and Reetuparna Das. Compute-capable block rams for efficient deep learning acceleration on fpgas. In *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 88–96. IEEE, 2021.

[216] Xiaowei Wang, Jiecao Yu, Charles Augustine, Ravi Iyer, and Reetuparna Das. Bit prudent in-cache acceleration of deep convolutional neural networks. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 81–93. IEEE, 2019.

[217] Yitu Wang, Zhenhua Zhu, Fan Chen, Mingyuan Ma, Guohao Dai, Yu Wang, Hai Li, and Yiran Chen. Rerec: In-reram acceleration with access-aware mapping for personalized recommendation. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2021.

[218] Ziheng Wang. Sparsert: Accelerating unstructured sparsity on gpus for deep learning inference. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, pages 31–42, 2020.

[219] Paul N Whatmough, Chuteng Zhou, Patrick Hansen, Shreyas Kolala Venkataramanaiah, Jae-sun Seo, and Matthew Mattina. Fixynn: Efficient hardware for mobile computer vision via transfer learning. *arXiv preprint arXiv:1902.11128*, 2019.

[220] Mark Wilkening, Udit Gupta, Samuel Hsia, Caroline Trippel, Carole-Jean Wu, David Brooks, and Gu-Yeon Wei. Recssd: Near data processing for solid state drive based recommendation inference extended abstract.

[221] Carole-Jean Wu, David Brooks, Kevin Chen, Douglas Chen, Sy Choudhury, Marat Dukhan, Kim Hazelwood, Eldad Isaac, Yangqing Jia, Bill Jia, et al. Machine learning at facebook: Understanding inference at the edge. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 331–344. IEEE, 2019.

[222] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[223] Minhui Xie, Youyou Lu, Jiazhen Lin, Qing Wang, Jian Gao, Kai Ren, and Jiwu Shu. Fleche: an efficient gpu embedding cache for personalized recommendations. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 402–416, 2022.

[224] Hao Ye, Geoffrey Ye Li, and Biing-Hwang Juang. Power of deep learning for channel estimation and signal detection in ofdm systems. *IEEE Wireless Communications Letters*, 7(1):114–117, 2017.

[225] Praveen Yedlapalli, Nachiappan Chidambaram Nachiappan, Niranjan Soundararajan, Anand Sivasubramaniam, Mahmut T Kandemir, and Chita R Das. Short-circuiting memory traffic in handheld platforms. In *2015 Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 166–177, 2014.

[226] Greg Yeric. Moore's law at 50: Are we planning for retirement? In *2015 IEEE International Electron Devices Meeting (IEDM)*, pages 1–1. IEEE, 2015.

[227] Yafeng Yin, Lei Xie, Yuanyuan Fan, and Sanglu Lu. Tracking human motions in photographing: A context-aware energy-saving scheme for smart phones. *ACM Transactions on Sensor Networks (TOSN)*, 13(4):1–37, 2017.

[228] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Proceedings of the 27th International Conference on Neural Information Processing Systems-Volume 2*, pages 3320–3328, 2014.

[229] Moustafa Youssef, Mohamed Amir Yosef, and Mohamed El-Derini. Gac: energy-efficient hybrid gps-accelerometer-compass gsm localization. In *IEEE Global Telecommunications Conference GLOBECOM*, pages 1–5. IEEE, 2010.

[230] Jiecao Yu, Andrew Lukefahr, David Palframan, Ganesh Dasika, Reetuparna Das, and Scott Mahlke. Scalpel: Customizing dnn pruning to the underlying hardware parallelism. *ACM SIGARCH Computer Architecture News*, 45(2):548–560, 2017.

[231] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2018.

[232] Haibo Zhang, Prasanna Venkatesh Rengasamy, Shulin Zhao, Nachiappan Chidambaram Nachiappan, Anand Sivasubramaniam, Mahmut T Kandemir, Ravi Iyer, and Chita R Das. Race-to-sleep+ content caching+ display caching: A recipe for energy-efficient video streaming on handhelds. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 517–531, 2017.

[233] Haibo Zhang, Shulin Zhao, Ashutosh Pattnaik, Mahmut T Kandemir, Anand Sivasubramaniam, and Chita R Das. Distilling the essence of raw video to reduce memory usage and energy at edge devices. In *Proceedings of the 52nd Annual IEEE/ACM international symposium on microarchitecture*, pages 657–669, 2019.

[234] Jiaqi Zhang, Xiangru Chen, Mingcong Song, and Tao Li. Eager pruning: algorithm and architecture support for fast training of deep neural networks. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pages 292–303. IEEE, 2019.

[235] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. Cambricon-x: An accelerator for sparse neural networks. In *Microarchitecture (MICRO), 49th International Symposium on*, pages 1–12. IEEE, 2016.

[236] Xingyao Zhang, Chenhao Xie, Jing Wang, Weidong Zhang, and Xin Fu. Towards memory friendly long-short term memory networks (lstms) on mobile gpus. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 162–174. IEEE, 2018.

[237] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1059–1068, 2018.

[238] Xuda Zhou, Zidong Du, Qi Guo, Shaoli Liu, Chengsi Liu, Chao Wang, Xuehai Zhou, Ling Li, Tianshi Chen, and Yunji Chen. Cambricon-s: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 15–28. IEEE, 2018.

[239] Yuhao Zhu and Vijay Janapa Reddi. Webcore: Architectural support for mobileweb browsing. *ACM SIGARCH Computer Architecture News*, 42(3):541–552, 2014.

[240] Yuhao Zhu and Vijay Janapa Reddi. Optimizing general-purpose cpus for energy-efficient mobile web computing. *ACM Transactions on Computer Systems (TOCS)*, 35(1):1–31, 2017.

[241] Yuhao Zhu, Anand Samajdar, Matthew Mattina, and Paul Whatmough. Euphrates: Algorithm-soc co-design for low-power mobile continuous vision. *arXiv preprint arXiv:1803.11232*, 2018.

[242] Yuhao Zhu, Aditya Srikanth, Jingwen Leng, and Vijay Janapa Reddi. Exploiting webpage characteristics for energy-efficient mobile web browsing. *IEEE Computer Architecture Letters*, 13(1):33–36, 2012.