Application Aware Data Management in Edge Computing

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Nikhil Sreekumar

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Dr. Abhishek Chandra (Advisor), Dr. Jon Weissman (Co-advisor)

February, 2026

Acknowledgements

I would like to express my deepest gratitude to my advisors, Dr. Abhishek Chandra and Dr. Jon Weissman, for their unwavering support throughout my doctoral journey. There were moments when I questioned the relevance of my work and felt ready to pivot, but their constant encouragement gave me the resolve to persevere. The contributions within this thesis are a direct result of their guidance and faith in my research.

My entry into the world of research was shaped by Lei Huang, Zhiying Liang and Cody Perakslis, who provided invaluable support during my first project. I am also grateful to Joel Wolfrath, Dhruv Kumar and Yuanli Wang, who helped me navigate and discover new research domains. Each of them played a vital role in my professional and technical development.

I want to thank Rankyung Hong for being a constant source of positivity in the lab. Her encouragement was often the highlight of my day. I also thank Yixuan Wang for her partnership during our coursework and research. Her unique perspectives became increasingly valuable as my work matured. Special thanks to Mitchell Terrell, whose insights during our one-on-one discussions and team meetings were instrumental in refining my research. I am truly grateful for the time and mentorship he dedicated to me.

I also wish to thank the students I have had the privilege to mentor: Vayam Agarwal, Diane Li, Hafsa Abdirahman, Priya Eturi and Steven Moore. You taught me the profound lesson that knowledge can be imparted by anyone, regardless of age or experience. I will always be thankful for what I learned from you.

I am deeply grateful to Dr. Jaideep Srivastava for his support during the final stages of my research and for the opportunity to collaborate on multiple research proposals. A special thanks goes to Lokendra Chauhan, who has been pivotal in helping me bridge

the gap between academic research and real-world application. I am grateful for his confidence in me to build a team for his firm, an experience that exposed me to the complexities of commercialization and compliance.

To my roommate of four years, Shaoming Xu, thank you for the many group activities that allowed me to explore and build lasting friendships outside the lab. Our philosophical discussions and your unique view of life provided much needed food for thought and helped me maintain my perspective.

I am also incredibly thankful to Wilson, Shyne, Rohini, Naveen, Preethi, Lakshmy, Priti, Yana, Anita, Katie, Michael, Anupriya and Sreehari. Your friendship and support throughout this journey made the challenges of the PhD much easier to bear.

To my parents, thank you for your endless support. You encouraged me to keep moving forward no matter the obstacles. I also thank my brother and sister-in-law for their instrumental guidance. Your advice on life and career paths significantly influenced my research trajectory.

Finally, to my beloved Mahima, who has been my greatest source of strength since we began our journey together four years ago. Beyond being a fellow researcher who understood the challenges of this path, you were the force that kept me going. Your unwavering belief in me, even when I doubted myself, is the reason I am standing here today. Thank you for being the cornerstone of my life throughout this journey.

Lastly, I thank the Almighty for all the blessings bestowed upon me throughout this path.

Abstract

Edge computing confronts a massive data surge from IoT devices, crucial for emerging low-latency applications like AR/VR, autonomous vehicles, and real-time healthcare. However, the edge's inherent limitations, scarce resources, network instability and system heterogeneity, mean that traditional cloud-centric data management strategies cannot be effectively applied 'out of the box', necessitating edge-specific approaches. This dissertation posits that **application awareness** is fundamental to overcoming these challenges, requiring strategies adapted to the unique characteristics and goals of edge applications. We introduce and evaluate four complementary techniques: 1) **Cargo**, a fault-tolerant, performance-aware storage layer integrated into the Armada edge framework, enabling stateful applications on volatile resources. 2) A comparative analysis of data placement strategies, demonstrating the scalability benefits of a novel **spatial-awareness heuristic** for fast server selection in dense environments. 3) **ASTRA**, an intelligent prefetching system for mobile AR that leverages object associations and user proximity, to achieve high cache hit rates. 4) **Viveka**, an activity-aware framework for wearables that co-optimizes sensor selection and sampling rates, yielding high energy savings with minimal accuracy impact. Collectively, these advancements provide a holistic approach to building efficient, responsive, and context-aware data management systems for the heterogeneous edge.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Edge Computing and Data

The exponential proliferation of Internet of Things (IoT) devices, sensors and mobile technologies have generated an unprecedented *data deluge* at the edge of the network. There is a growing demand for applications characterized by ultra-low latency service and real time responsiveness, such as augmented/virtual reality, autonomous vehicles, healthcare, industrial automation and smart city infrastructure. It is estimated that, by 2025, 75% of the enterprise data generated will originate from outside the traditional data center or clouds [2]. This deluge has marked a paradigm shift from centralized cloud computing models towards distributed edge computing infrastructures.

Edge computing model is characterized by the placement of computation, storage and networking resources closer to users and source locations of data generation. It includes a wide spectrum of infrastructure such as end-user devices (smartphones, vehicles, drones), internet gateways in industrial structure/smart homes, on-premise servers, cloudlets [3], cellular base stations, roadside units and so on. Edge computing acts as a complementary paradigm to cloud computing rather than a replacement. In this synergistic setting, the edge handles tasks requiring low latency service, real time processing, local data handling and bandwidth optimization, while the cloud provides large scale computation, elastic storage, complex analytics and model training capabilities.

The shift towards edge computing introduces multiple challenges for data management driven by the distinct characteristics of edge data and the limitations of edge

environments. The main characteristics of data at the edge are

1. Massive volume: The handling of the unprecedented data deluge at the network edge from different edge endpoints solely though centralized cloud infrastructure is impractical and economically prohibitive [4].

2. High generation rate: The data is continuously generated and requires real-time processing mostly for the information to be valuable. For instance, autonomous driving depends on immediate inferences from data generated for safe operation.

3. Heterogeneous data: The data at the edge originates from multiple sources (sensors, audio/video devices and so on), each with a different format, contextual relevance and transferred via different communication protocols. The aggregation of these data streams poses a considerable challenge.

In addition to the data characteristics, the management of data at the edge is further challenging due to the following inherent edge constraints.

1. Limited resources: Edge devices (smartphones, tablets, drones) and nodes (servers, gateways) are limited in compute, storage and energy capacity, necessitating the development of data management strategies that minimizes resource consumption.

2. Network constraints: Edge devices are connected to the servers via limited or variable network bandwidth. Sometimes, continuous connectivity may be costly or unavailable. Given the intermittent connectivity nature of network, it is necessary to build a robust and reliable data management approach.

3. Heterogeneity: The edge environment consists of diverse hardware platforms, operating systems, software stacks and communication protocols. This diversity introduces challenges for universally applicable data management policies for the edge.

This combination of diverse, real-time data streams, coupled with severe resource constraints creates a data management landscape at the edge that is fundamentally distinct from the elastic environment of centralized clouds. This necessitates a paradigm shift towards more intelligent, adaptive, and context-aware data management strategies [5].

1.2 Application-Awareness and Rationale

Application-aware data management in edge computing is a strategy that is explicitly tailored and dynamically adapted based on specific characteristics, requirements and context of applications utilizing edge data. This is in contrast with the application agnostic approaches that apply uniform policies across all data flows, leading to sub-optimal performance in the heterogenous, resource constrained edge. This awareness is not an enhancement, rather a fundamental requirement driven by several key factors:

1. Addressing edge constraints: A generic, application-agnostic data management strategy leads to inefficiencies, wasting limited resources on non-critical data or failing to prioritize latency-sensitive tasks. Application awareness allows for the intelligent allocation and utilization of these limited resources based on actual need [6, 7].

2. Optimizing resource utilization: Data management strategies can be tailored to reduce data movement, storage footprint and computational load based on application awareness. For example, reducing redundant sensor data to save storage and bandwidth in monitoring applications, placing frequently accessed item for an application in a nearby edge node.

3. Enhanced application performance: Edge applications span a wide spectrum from critical low latency service in autonomous vehicles to latency tolerant soil feature updates in smart agriculture. Depending on the application's critical nature, resource prioritization and data path optimizations could be necessary. This ensure a wide range of applications can co-exist at the edge while ensuring QoS guarantees [8, 9].

4. Enabling killer applications: The applications that would draw out the full potential of edge computing are called killer applications, for example tactile internet, AR/VR, autonomous vehicles, robotics to name a few. It is not feasible to achieve the highly optimized, low latency services for these unless application specific requirements are involved in the task and data handling decision process [10, 11].

Application awareness extends beyond purely latency and bandwidth, but requires a holistic view across the entire data lifecycle. It should inform decisions from the point

of data generation through processing, storage/caching and transmission. This perspective ensures that the data management policies are coherent and optimized throughout according to application requirements.

1.3 Strategies for Application-Aware Data Management

As mentioned in the previous section, the awareness must be a part of the entire data lifecycle. We will go through them one by one and finally focus on two stages for the thesis contributions.

1.3.1 Data filtering, aggregation and prioritization

Edge devices generate vast amounts of data, presenting a significant management challenge due to the sheer volume. Intelligent data filtering and aggregation techniques are required to minimize the data needs processing, storage, and transmission [12]. This can involve selectively discarding irrelevant or redundant data based on application requirements. Furthermore, applications often do not need raw data, but rather summaries aggregated periodically or in response to specific events.

For example, in human activity recognition, inferring each activity does not necessarily require the same sample rate from all sensors [13]. Also, it might be that not all sensors are needed to infer a specific activity [14, 15]. In augmented reality, an edge device may use tracking and frame detection mechanisms to determine if the current frame is a key frame. To avoid unnecessary bandwidth usage, only key frames should be sent to an edge server for further processing.

In an edge network, not all data flows are equally time-sensitive. Application-aware data prioritization involves differentiating and managing network traffic based on the critical nature and QoS requirements of the application. The goal is to ensure that data packets belonging to critical or latency-sensitive applications receive preferential treatment over those from less critical or latency-tolerant ones. This prioritization is especially crucial during periods of network congestion or resource contention. Data prioritization can be implemented using various approaches, including network-level QoS, latency awareness [16], and load-aware strategies [17].

For example, consider a shared edge network used by two applications: one is a

multi-player game and the other is an emergency response system. During periods of congestion, it may be necessary to prioritize emergency response packets over game packets.

The benefits of filtering and aggregation are significant: they reduce network bandwidth consumption, lower data transmission costs and decrease storage requirements on edge nodes. Furthermore, these techniques reduce power consumption and the processing load on downstream systems. These reductions allow the edge infrastructure to handle large volumes of data efficiently. The primary benefit of prioritization is that it ensures critical and time-sensitive applications meet their performance targets even in congested or resource-constrained edge networks. It also improves the overall user experience for prioritized services.

1.3.2 Data placement and caching

The objective of application-aware data placement and caching strategies is to identify strategic server locations for data replicas, ensuring optimized access for the applications that require them. Factors influencing these placement decisions include the popularity of data across user groups [18], characteristic access patterns (such as sequential, random, read-heavy, or write-heavy), application latency requirements, user mobility, and prevailing resource constraints [7, 19].

In vehicular computing, for instance, placing relevant traffic data on nearby Roadside Units is preferable to using edge servers covering larger areas due to the critical low-latency requirements of autonomous vehicles. Similarly, in AR-enabled multiplayer games, specific AR assets might only appear when certain in-game conditions are met. Consequently, data for these assets could be strategically placed on edge servers near locations with high user density where players are likely to meet those conditions.

The primary benefits of application aware data placement are low latency data access, improved responsiveness and reduced backhaul traffic.

1.4 Challenges

In the previous section, we identified different approaches for application-aware data management. The main challenges associated with implementing each approach are

discussed below.

Data filtering, aggregation and prioritization:

1. Data relevance and criticality: Defining which data is relevant, redundant, or critical for a specific application's purpose is complex and highly context-dependent. Filtering excessively can hinder application insights, while retaining too much raw data consumes limited resources unnecessarily.

2. Resource constraints: Edge devices operate with limited processing power, memory, and energy budgets. Implementing sophisticated filtering or aggregation algorithms locally can be computationally demanding, potentially exceeding device capabilities or rapidly depleting their power reserves.

3. Data heterogeneity: Edge data originates from diverse sources using varying formats and protocols. Applying consistent filtering or aggregation rules across these heterogeneous data streams and effectively integrating the results presents a significant challenge.

4. Data reliability and effectiveness: Data gathered at the edge can be unreliable due to sensor limitations or unpredictable environmental factors. Filtering or aggregating noisy, incomplete or inaccurate data may lead to flawed analyses and incorrect downstream decisions.

5. Computational overhead: While filtering and aggregation techniques reduce data volume to save bandwidth and downstream processing, they inherently introduce computational overhead. For time-sensitive applications, this overhead might be unacceptable, necessitating a careful trade-off between processing latency and data reduction benefits.

6. Information loss: The process of filtering or summarizing data can lead to the loss of potentially valuable information present in the raw data. This loss could negatively impact subsequent task analysis or the performance of machine learning models.

7. Algorithm design: Designing and deploying filtering/aggregation algorithms that are both adaptive and lightweight enough to execute efficiently on resource-constrained

edge devices is a non-trivial task.

Data placement/caching:

1. Resource constraints: Edge nodes possess limited storage capacity, processing power, network bandwidth, and energy budgets. Efficient data placement and caching must carefully consider these limitations across potentially heterogeneous edge infrastructure. Effectively managing finite cache space remains a key challenge.

2. Balancing trade-offs: Application requirements necessitate balancing various trade-offs in placement and caching strategies. For example, maximizing the cache hit ratio might favor caching highly popular content, whereas minimizing latency for time-sensitive applications could require caching less popular but critical data closer to users. Inherent trade-offs also exist between latency, cost, scalability, and scheduling depending on the chosen method.

3. Dynamic and heterogeneous environments: Edge environments are inherently dynamic, featuring changing network conditions, user mobility, and fluctuating content popularity. Furthermore, edge nodes are heterogeneous, varying in capacity and capabilities. A significant challenge is sesigning placement and caching strategies that adapt effectively to these dynamics and heterogeneity.

4. Complex decision making: Optimal placement and caching involve navigating multiple complex decisions: what data to place/cache, where to place/cache it, and when to replace existing entries. These decisions must also be informed by the application's scalability requirements.

5. Data consistency: Depending on application needs, data must remain consistent across replicas and caches. Scalability requirements at the edge might necessitate a higher number of replicas compared to traditional cloud environments, potentially requiring consistency policies specifically tailored for the edge.

1.5 Thesis Contributions

This thesis develops and evaluates novel strategies for application-aware data management, focusing on two representative and demanding application domains: Augmented Reality and Human Activity Recognition. The main contributions are organized as follows::

1. Chapter 2 addresses the need for reliable persistent storage in volatile edge environments by introducing **Cargo**, the storage layer for the Armada framework. Cargo decouples storage from compute, utilizing **compute-proximity based selection** and **performance probing** to place data replicas on optimal edge nodes near consuming tasks. It enhances resilience through automated replication and seamless failover, demonstrating that leveraging managed **volunteer** resources provides a significantly lower-latency fallback than reverting to the cloud.

2. Chapter 3 tackles the optimal placement of data replicas on edge servers considering latency, load and capacity constraints. It formulates the problem as an MINLP and compares three heuristic strategies: distance-based, latency-based and **a novel spatial-awareness-based approach using R-Tree pruning**. Evaluations show that while latency-based selection is effective, the **spatial heuristic offers superior scalability** by drastically reducing server selection time in dense environments without significant performance compromise, making it ideal for large-scale deployments.

3. Chapter 4 introduces **ASTRA**, a novel application-aware prefetching framework designed to reduce latency in Mobile Augmented Reality (MAR). ASTRA enhances edge cache performance by integrating **object associations** (mined from interactions), **temporal relevance** (via an association factor) and **spatial proximity** (using a lazy fetching strategy). It achieves up to 35% higher cache hit rates and reduces end-to-end latency by up to 14% compared to baselines. An adaptive minimum support tuning mechanism further enhances performance by up to 10% under dynamic workloads.

4. Chapter 5 addresses energy efficiency at the data source, proposing **Viveka**, an activity-aware framework for smart wearables. Viveka **co-optimizes sensor selection and sampling rates** dynamically based on inferred user activity. It employs **Permutation Feature Importance (PFI)** to identify activity-specific sensor subsets and the **Nyquist-Shannon theorem** to determine minimal per-sensor sampling rates, managed by robust on-device logic. Evaluations demonstrate Viveka achieves significant **energy savings (up to 62%) and data reduction (up to 74%)** compared to baselines while maintaining high HAR accuracy.

5. Chapter 6 synthesizes the findings from the preceding chapters, discusses the broader implications of application-aware data management at the edge, acknowledges limitations and outlines promising directions for future research.

Chapter 2

Cargo: Application performance aware storage in Edge computing

2.1 Introduction

Edge environments are heterogeneous in terms of network, compute, storage and load. During storage/data placement decisions, simply selecting the geographically closest node for an application doesn't guarantee low latency. Therefore, effective placement decisions must also consider factors such as network congestion, server load and hardware performance. This necessitates an application performance aware approach to storage selection for real time low latency services.

Many of the existing methods focus on task/service placement [20–22] without considering the storage location for the applications and how the data retrieval will affect the ened-to-end latency. Strategic placement of data and its replicas is essential for mitigating data transfer latency, a primary bottleneck in end-to-end performance. We take the first step towards this exploration in this chapter and show further improvements in the Chapter 3.

To address these challenges, this chapter introduces Cargo, the persistent storage layer for Armada, a framework designed for dense, heterogeneous edge environments. Armada itself provides elasticity for compute offloading through a novel two-step task scheduling approach [23, 24]. However, to support the vast number of stateful and data-intensive edge applications, a dedicated storage solution is required. Cargo is designed

explicitly to fill this role. From an application's viewpoint, it acts as a reliable persistence service located at the edge, providing a low-latency alternative to accessing data from the cloud and resilience against the high churn of volunteer compute resources.

The main contributions in this work are:

1. Design and implementation of the Armada storage layer, Cargo, which employs a compute-proximity-based selection strategy to place data on edge servers for low-latency access.
2. Seamless switching across replicas to provide fault tolerance and timely service.
3. Support for strong and eventual consistency depending on application requirements.

2.2 Armada Overview

In this section, we describe the heterogeneous edge-dense environment and give an overview of Armada design goals and system architecture. Then we discuss the application type that Armada supports from a storage perspective.

2.2.1 Heterogeneous Edge-Dense Environment

Low-latency communication channels between users and edge servers, known as logical proximity, are typically provided by dedicated infrastructure like LANs, on-premise networks, or 5G. However, these special-purpose resources are often constrained in availability and scalability. In Figure 2.1, we show that nearby general-purpose resources in heterogeneous WAN environments (Edge-tier-2) can also provide low-latency benefits when Edge-tier-1 resources are not available or overloaded. We include both dedicated local public servers and volatile volunteer resources in Edge-tier-2 to enlarge the edge presence. Therefore, the resource limitation on edge can be resolved with the help of abundant volunteer edge nodes densely distributed around users, namely *edge-dense* environments.

The heterogeneity of Edge-tier-2 resources is twofold. First, connections from users to edge servers in WAN environments are highly diverse in terms of local ISPs and underlying networking infrastructure. Based on how users connect to the network, the

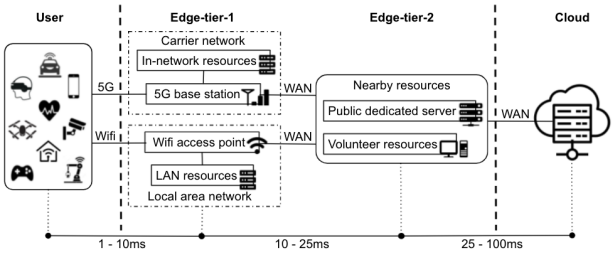


Figure 2.1: RTT latency in heterogeneous edge-dense environment

actual number of routing hops and latency performance to the same edge server can highly diverge. Second, accessible compute resources present in nearby areas come from multiple providers and individuals. The heterogeneous capacity and hardware can lead to different processing performance, which is on the critical path of user requests and thus affects the end-to-end latency. Volunteer resources will amplify such heterogeneity by introducing more edge access points and increasing the system entropy.

2.2.2 Design Goals

Armada is designed with the following goals in mind, with this chapter focusing specifically on the design and implementation of the in-situ edge storage layer:

- **Support for low-latency computation offloading at scale with densely distributed edge resources:** While one edge server is limited by its capacity, many loosely coupled but densely distributed edge nodes can coordinate with each other to provision nearby users at scale. Armada is designed to manage resource-constrained but abundantly distributed edge nodes to support scalable low-latency computation offloading. As a result, applications deployed on Armada are able to automatically scale and obtain more resources in a specific region if more users are present.

- **Locality-based service deployment:** Service deployment should be based on fine-grained geographical specifications to reduce networking latency. Multiple replicas [1] of the service should be deployed on different edge nodes to guarantee edge availability and capacity in specified regions. Changes to currently active

[1] We use the term service replica and task interchangeably in this paper.

users should also dynamically guide the service placement to fit the real-time user distribution. Furthermore, new service deployment should be optimized for short startup time to start serving users in a timely manner.

- **Performance-aware service selection in heterogeneous environments:** Geographical proximity is not strictly equivalent to low RTT latency. Multiple factors together determine the edge performance including network/compute resource heterogeneity and availability. Given a list of nearby edge nodes running replicas of the application service, Armada should identify the best-performing edge access point for each user to offload the computation. This edge selection process should also handle the load balancing for all users to achieve overall lower latency.

- **Ease of use:** Armada interfaces should be easy to use for both application developers and resource contributors. In particular, developers should use Armada SDK with minimum code modifications to their applications for deployment. Moreover, resource contributors should be able to register their nodes quickly with lightweight components and isolated runtime.

- **Fault tolerance:** Armada must ensure the fault tolerance for Armada users in the presence of high node churn due to volatile, unreliable and unpredictable volunteer resources. Armada users must be guaranteed continuous service and experience zero downtime upon node failure or node leaving.

- **In-situ edge storage:** Armada should provide a native storage layer on the edge [25] to support low-latency data access. The storage layer, Cargo, should be reliable and independent from the volatile compute layer to persist the data for stateful and data-intensive applications. Also, flexible duplication and consistency policies should be supported for different application requirements.

2.2.3 Armada Architecture

Figure 2.2 shows the Armada system architecture. Armada consists of geo-distributed nodes that donate their compute and/or storage resources, along with a set of global

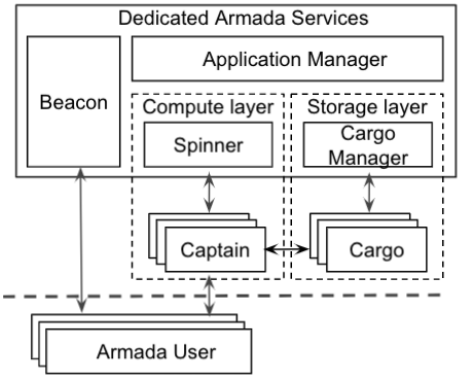


Figure 2.2: Armada system architecture

and central services hosted on dedicated, stable nodes. Both Armada system components and Armada-hosted applications are encapsulated in Docker containers for ease of use and fast deployment. Docker itself provides a lightweight, isolated runtime and abstractions over underlying resources for edge nodes, which is a good option for shipping the code easily to volunteer-based heterogeneous environments. Armada resources and services together constitute the following major components:

- **Beacon:** Beacon is the global entry point for all interactions with Armada central services. It will forward requests to corresponding handler components, including application deployment requests, user connection requests and resource registration requests.

- **Application Manager:** Application manager maintains the states of submitted applications in Armada and manages the application lifecycle. It globally controls, operates, and monitors all application tasks running on different edge nodes, and processes initial user connecting requests. It also handles auto-scaling based on real-time user demand.

- **Compute Layer:** Compute layer manages dedicated and volunteer compute resources in Armada. It includes Spinner, the compute resource manager and Captain, the compute node. The Spinner handles compute node registration, health check and resource allocation for task deployment requests sent by the Application manager. The Captain manages the local heterogeneous resources through

the Docker engine API and processes user workloads.

- **Storage Layer:** Storage layer manages dedicated and volunteer storage resources in Armada. It includes Cargo manager, the storage resource manager and Cargo, the storage node. The Cargo manager handles storage node registration, health check, maintains metadata and executes storage policies for data-dependent applications. The Cargo manages the local heterogeneous storage resources using the Docker volume and persists data on the edge supporting low-latency access for nearby users. The design, implementation, and evaluation of this Storage Layer, particularly the Cargo components, form the core contribution of this chapter.

2.2.4 Armada Applications

Armada applications are long-running edge services using Armada resources for low-latency computation offloading. It includes a server-side program submitted to Armada for application-specific processing, and a client-side program used by application users to discover the service and offload computations. Armada deploys multiple replicas of the server-side program (tasks) to guarantee availability and scalability. Moreover, the client-side program uses Armada SDK to help application users locate the nearby service access points and establish direct communication channels. In Armada, we focus on the scenario where application users are co-located with the processing data, such as AR users sending out video streams for real-time processing. However, we also support external data upload from other data sources to the Armada storage layer, providing low-latency data access for running services.

In Armada, volunteer resources are assumed to be unstable, volatile, and dynamic, with high node churn in heterogeneous environments. The guarantee on immediate recovery and continuous services upon node failure requires that application clients immediately switch connections to other service replicas and continue processing without waiting for failed node recovery. Therefore, no hard states or dependencies of the users are allowed to be maintained on the server-side for Armada applications. Application developers should either modify the application to maintain hard states and execution contexts on the client-side or use the Armada storage layer through Armada storage SDK to persist the data with minimized latency overhead.

This fundamental constraint, the inability to maintain state on volatile compute nodes, directly motivates the need for a reliable, independent persistence service. The Armada storage layer, Cargo, is designed precisely to be this service, enabling the development of robust stateful applications by providing a stable, low-latency data backend.

2.2.5 Armada Storage Layer

The storage layer, Cargo, is fundamentally designed around the principle of application awareness to meet the diverse and demanding requirements of stateful edge applications. Unlike generic storage systems, Cargo's behavior, from initial configuration to real-time operation, is directly tailored by application-specific needs:

- Application-Specific Configuration: Recognizing that edge applications have diverse persistence and consistency needs, Cargo requires applications to explicitly declare their requirements (capacity, desired consistency level, data source location) during deployment. This a priori knowledge allows Cargo Manager to provision resources and configure replication strategies appropriately from the start.

- Latency-Optimized Placement: To satisfy stringent application latency requirements, Cargo employs a proximity-focused placement strategy. The Cargo Manager aims to co-locate data replicas near the consuming compute tasks (Captains), minimizing network distance. Furthermore, placement and auto-scaling decisions adapt based on observed application demand and user distribution, ensuring resources are allocated where the application needs them most.

- Dynamic Performance Optimization for Applications: Cargo ensures the application task always accesses the lowest latency available data replica. Captain nodes, acting on behalf of the application, use Armada's 2-step discovery and performance probing mechanism to dynamically identify and connect to the best performing nearby Cargo node. This dynamic connections allows adapting to real-time network conditions to maintain optimal application responsiveness.

- Simplified Application Interaction: Applications interact with the storage layer via a straightforward SDK. This abstracts away the complexity of replica management

and connection handling, allowing developers to easily integrate persistent storage with minimal code changes.

- Application defined Consistency: Cargo acknowledges that different edge applications have varying tolerances for data staleness versus access latency. It allows applications to select their desired consistency model (e.g., strong or eventual) during registration. This enables developers to make explicit, application-specific trade-offs between data freshness and performance, rather than imposing a one-size-fits-all policy.

- Data Resilience for Applications: The storage layer provides fault tolerance through data replication across multiple Cargo nodes. If a Cargo node fails, the application task can automatically switch to another available replica, ensuring continuous access to its persistent data with minimal disruption.

The storage layer consists of two components: Cargo Manager, the storage resource manager and Cargos, the geo-distributed storage nodes.

Cargo Manager

Cargo manager manages edge storage resources in Armada. Table 2.1 shows Cargo manager interfaces: for Cargos to join and report status, Application manager to allocate storage resources, and Captains to discover nearby data access points. Cargo manager also spawns data replicas to guarantee fault-tolerance and low-latency data access for geo-distributed services. Data persistence is achieved on edge with redundant data replicas and flexible data consistency policies. The three main modules of Cargo manager are described as follows:

Storage registration: Application manager sends the *Store_Register* request to the Cargo manager during the service deployment phase if the application requires persistent edge storage. The *Store_Register* request contains the service identifier, capacity requirement for each data replica, consistency policy, and the data source for original data uploading. The initial three Cargos are selected based on their geo-proximity to the service's target deployment region and their available capacity. The Cargo selection is based on the application's specified consistency policy and the geographic distribution of its anticipated compute tasks.

Interface	Input/Output	Description
Cargo_Join	Cargo_Metadata/ Status	A new Cargo registers itself into the system.
Cargo_Update	Cargo_Updates/	Cargo sends heartbeats to Cargo manager reporting status updates.
Store_Register	Storage_Req/ Status	Application manager registers storage capacity for an edge service.
Cargo_Discover	Captain_Info/ Status	Captain queries nearby data access points

Table 2.1: Cargo manager interfaces

Data access point selection: Cargo manager maintains the metadata and states of all data replicas for an edge service. After the storage registration, Captain sends *Cargo_Discover* requests during the task deployment phase to help tasks find nearby data access points. A similar *2-step* approach in service selection is applied to overcome the network heterogeneity and locate the best-performing data access point. First, a candidate list is generated by the Cargo manager based on the geo-proximity between the Captain and Cargos holding the data replicas. Optional factors like network affiliation can also be specified to help rank the candidates. Second, the Captain performs data access probing by measuring the round-trip latency for a representative dummy workload to each candidate Cargo node. This allows it to identify the fastest access point under current network conditions, rather than relying on static geo-proximity alone. This ensures that each application task connects to a replica that minimizes its specific data access latency under current conditions.

Storage auto-scaling: Initially a service is allocated three storage replicas. When more service replicas are spawned to satisfy higher user demand and wider user distribution, the storage layer should also adaptively scale to guarantee low-latency data access aligned with the application's evolving geographic footprint and performance needs.. We employ the similar idea applied in the service auto-scaling process. When new service replicas are deployed, the Cargo manager asynchronously creates new data replicas on geo-proximate Cargos to the services. Since more replicas lead to higher

resource usage and data consistency overhead, the Cargo manager collects the data access probing feedback from Captains to evaluate the need to spawn new data replicas carefully. This evaluation is triggered when a new compute task (Captain) is deployed in a region geographically distant from existing Cargo replicas or when existing tasks report consistently high latency to their current storage nodes.

Cargo nodes

Cargo is an edge storage node in Armada. It handles data I/O operations and propagation of updates to replicas based on the consistency level chosen by the application. For efficient update propagation, each Cargo node maintains connections to its peers in the replication chain. For instance, in a simple cascade, a node is aware of the next node to which it must forward updates. Table 2.2 describes the Armada storage SDK used by server-side application programs to interact with the storage layer. With Captains locating nearby data access points, Armada storage SDK helps tasks transparently communicate with nearby Cargos.

Function	Input / Output	Description
Init_Cargo	Cargo_App_ Metadata/ Status	Establish connection with a Cargo node
Write	Write_Data/ Write_Status	Write data to the Cargo node
Read	Read_Data/ Read_Status	Read data from Cargo node
Close_Cargo	_/Status	Close connection to Cargo node after use

Table 2.2: Armada storage SDK

2.3 Real-time Inference on Armada

We use face recognition, a critical building blocks in commonly used applications like augmented reality, cognitive assistance and security surveillance, for evaluating Cargo. It is computational-intensive and latency-sensitive, which require offloading the computation to powerful servers and obtaining processing results in a timely manner. The face recognition workload showcases the coordination between computing and storage

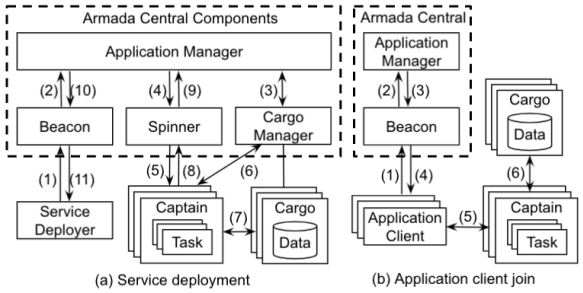


Figure 2.3: Face recognition workflow in Armada

layer when Armada application needs persistent edge storage. This workload is particularly well-suited for evaluating Cargo because it involves both frequent, low-latency read operations (matching detected faces against a database) and periodic write operations (adding new, unrecognized faces to the dataset), thereby testing the performance of the storage layer under a mixed read/write access pattern.

2.3.1 Face Recognition

The service deployment phase, illustrated in Figure 2.3(a), begins when a deployer submits the application along with its compute and storage requirements to Armada (steps 1-2). The Application Manager first coordinates with the Cargo Manager to provision the necessary storage (3). In response, the Cargo Manager selects three geographically appropriate Cargo nodes, allocates resources for data replicas, and instructs them to pull the initial pre-labeled face dataset from a specified source. Once the storage is prepared, the Application Manager deploys the compute tasks to the Compute Layer (4-5). A key step in this process is connecting the compute tasks to their data; the Captain nodes query the Cargo Manager to discover the list of nearby data access points (6). Using this candidate list, the tasks establish a direct connection to the optimal Cargo replica using the Armada storage SDK (7), completing the end-to-end setup. Finally, status updates are reported back to the deployer (8-11).

Figure 2.3 (b) shows the workflow when face recognition clients request the service. In step (1) - (4), clients first query the system for service access points, and then start sending video frames for face recognition in step (5). For any detected faces during the processing, tasks query data replicas in Cargos for face recognition [26] (6). The

Node	Processor	Processing
V1	Intel Core i7-9700, 8 cores	24ms
V2	Intel Core i7-2720, 6 cores	32ms
V3	Intel Core i9-8950HK, 6 cores	31ms
V4	Intel Core i5-8250U, 4 cores	45ms
V5	Intel Core i5-5250U, 2 cores	49ms
D6	Intel Xeon CPU E5-2620 v3, 24 cores	30ms X 4
Cloud	t2.large, 4 cores	34ms

Table 2.3: Real-world experiment hardware and frame processing time

Node	Type	Location	Processing
A	t2.2xlarge, 8 cores	City_A	23ms
B	t2.large, 4 cores	City_B	34ms
C	t2.small, 2 cores	City_C	58ms
Cloud	t2.large, 4 cores	Cloud	34ms

Table 2.4: Emulation experiment hardware and frame processing time

read requests send detected faces to Cargo searching for matched people, and the write requests insert new labeled faces into the persistent data store for future recognition.

2.4 Evaluation

We evaluate Armada in both real-world edge environments and emulation platforms in the cloud. The real-world experiment explores Armada performance in fine-grained small geographical areas (regions within a city). The emulation experiment explores wider-range geographical areas (regions across nearby cities). We use a face recognition workload to explore the storage layer performance when the persistent store is required.

2.4.1 Experimental Setup

In Table 2.3 and 2.4, we show the underlying hardware used for both real-world and emulation experiments. Note that the third column shows the processing time per frame for real-time face recognition application on these hardwares.

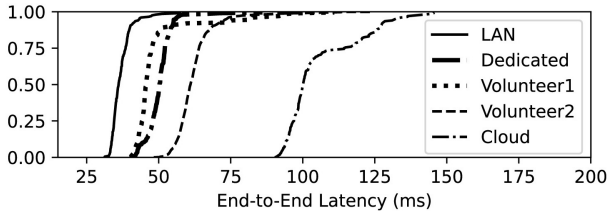


Figure 2.4: CDF of end-to-end latency for different servers

Real-world Environment

We set up the real-world experiment environment around our University campus. As shown in Table 2.3, a combination of both dedicated and volunteer resources is used. Volunteer nodes V1 - V5 are located within 5 miles of the campus, and a powerful University server D6 located on campus is considered the dedicated edge node.

While the dedicated node has more compute power and better network connectivity, volunteer nodes are set up with heterogeneous compute and networking performance contributed by actual volunteers around the campus. The dedicated node D6 can hold four service replicas in parallel, with each of them processing the video at 30ms/frame. Figure 2.4 demonstrates the performance heterogeneity in a real-world environment. It shows that for a given user, a well-connected volunteer node (e.g., V1, V2) can deliver a lower end-to-end latency than a more powerful but potentially less network-proximate dedicated node (D6). This finding directly motivates the need for a performance-aware selection mechanism over a simple resource-based or geo-proximity-based approach.

Emulation Environment

Due to physical limits, we use the emulation environment to explore Armada performance on a wider geographical scale. We use the network emulation platform Netropy [27] in AWS to emulate WAN connectivity for three nearby cities City_A, City_B and City_C that are about 100 - 150 miles away from each other. Three edge nodes A, B, C are located at three locations as shown in 2.4.

Baselines

We use volunteer, dedicated-edge-only and cloud scenarios for comparisons with Armada.

- **Volunteer**: It models a naive approach where clients connect to the geographically closest available volunteer node, without considering real-time network performance or server load.
- **Dedicated-edge-only**: In the dedicated-edge-only scenario, we assume that only limited dedicated edge resources are available, which is common in today's edge infrastructure deployment. As shown in Table 2.3, we use one powerful dedicated node as compared to 5 resource-constrained volunteer nodes to maintain a reasonable ratio of the availability of dedicated and volunteer resources. We show the benefits of exploiting volunteer resources by comparing them with the dedicated-edge-only scenario.
- **Cloud**: We show the cloud performance as the baseline compared to other scenarios. We use the closest AWS service region US East to deploy the services and assume that the cloud has unlimited scalability with increasing user demand.

2.4.2 Performance of Storage Layer

We use the face recognition workload to evaluate storage layer performance in the real-world experiment. In the following experiments, we focus on the communications between tasks and Cargos. Therefore we configure the *TopN* [23] to 1 to simplify the compute layer workflow. In this case, each application client only connects to one task.

We explore the effects of the Cargo selection strategy, storage fault tolerance and different consistency policies. The same set of resources described in Table 2.3 is used, with each one of them having 2GB persistent storage capacity. In addition, each data replica initially uploaded to Cargo contains 1000 labeled face descriptors [28] in the format of <ID (8 bytes), vector (128 * 8 bytes)> pairs. We focus on three workloads for evaluation:

Read-only workload: 1000 face images are used as the task input video frames for real-time recognition. The task processes each image, detects the face and generates

a unique face descriptor. Then the task queries Cargo to find the matched descriptor along with the face ID. The read latency includes the time to connect to the Cargo and query processing. The tasks do not buffer labeled faces locally to explore the Armada storage layer performance thoroughly.

Write-only workload: 1000 new face images are used as the task input video frames. We configure the task to detect faces and directly write new face descriptors with face IDs into the Cargo data replica. The write latency includes the time to connect to the Cargo and to perform the writing.

Read-followed-by-write workload: 1000 new face images are used as the task input video frames. For each image, the task first sends a read request to query the Cargo and then writes the new face descriptor into the Cargo when the read request cannot recognize the face.

Cargo selection

We explore the Cargo selection results using the read-only workload. Nodes V1, V2, D6 and Cloud are registered as four Cargos, and V3, V4 and V5 are used as Captains to run three face recognition tasks. We also configure three users co-located with three Captains for simplicity. Table 2.5 shows the Cargo selection result and pairwise read latency. We can see that the Cargo selection strategy can identify the environmental heterogeneity and select the best-performing data access point for each data-dependent task.

Task	Cargo_V1	Cargo_V2	Cargo_D6	Cloud
Task_V3	**21**	25	31	61
Task_V4	25	**23**	33	64
Task_V5	42	38	**18**	60

Table 2.5: Cargo selection

Storage fault tolerance

We demonstrate the storage fault tolerance behavior using the same experiment setup described in Section 2.4.2. In this experiment, we only focus on the read latency from Task_V5's perspective. Figure 2.5 shows that Task_V5 can immediately switch to the

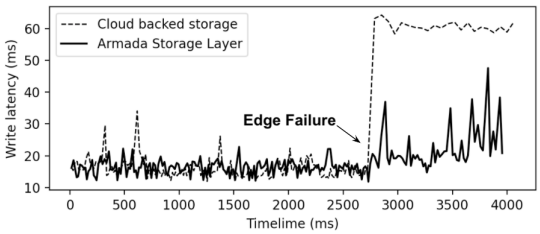


Figure 2.5: Continuous Cargo service on the edge

Cargo_V2 upon Cargo_D6 failure. The service is immediately restored at a low latency, in stark contrast to a cloud-backup approach, which would force a permanent shift to a high-latency cloud backend. This demonstrates Cargo's ability to maintain service continuity and low-latency performance at the edge even in the face of node failures.

Effect of Consistency

We run three workloads to explore the effect of different consistency policies in Armada. We also separate the performance for dedicated, volunteer edge resources and cloud to illustrate the benefits of exploiting volunteer resources for edge storage. We set up three configurations using dedicated Cargos, volunteer Cargos, and Cloud-located Cargos for both strong and eventual consistency scenarios. All edge nodes and users are loosely coupled with each other in real-world heterogeneous environments. As shown in Figure 2.6 and Figure 2.7, we record the data I/O latency with varying configurations, consistency policies, and workload types.

Figures 2.6 (a) and 2.7 (a) show that strong and eventual consistency have similar read latency since no data propagation is required for the read-only workload. Figures 2.6 (b) and 2.7 (b) show that the strong consistency for volunteer Cargos can cause higher latency than the cloud since volunteer nodes are loosely coupled, leading to high data propagation overhead. Similar to write-only workload, Figures 2.6 (c) and 2.7 (c) show that strong consistency has higher overhead caused by synchronized data propagation. Based on the above, volunteer Cargos in Armada exhibit similar performance compared to dedicated Cargos using eventual consistency. It also demonstrates the benefits of utilizing volunteer edge storage over the cloud for low-latency data access.

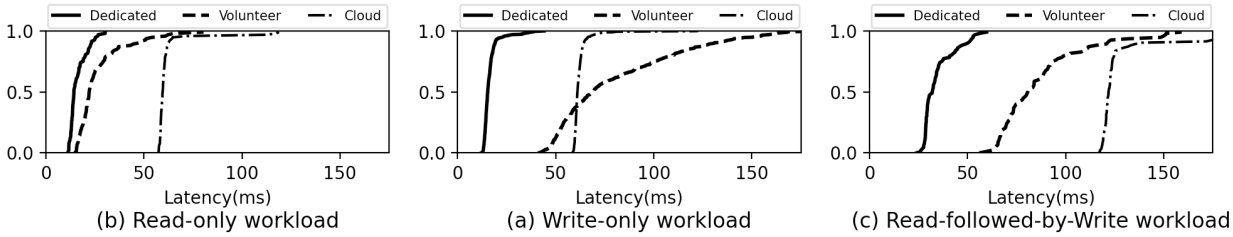


Figure 2.6: Read-Write latency for Strong Consistency

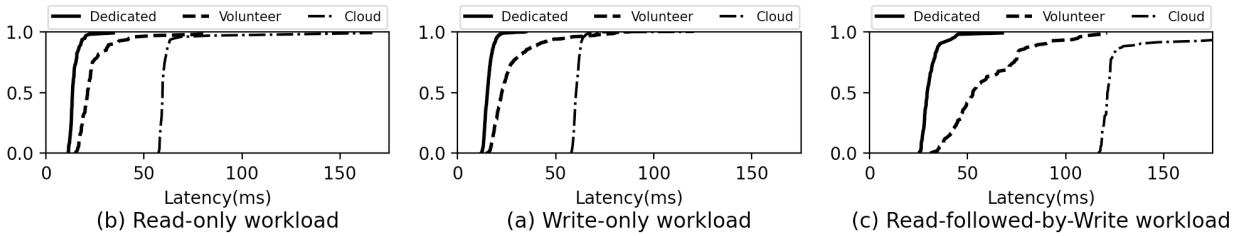


Figure 2.7: Read-Write latency for Eventual Consistency

The evaluation of consistency policies yields several key insights. First, for read-only workloads, both strong and eventual consistency deliver similar performance, as no data propagation is required. Second, for write-intensive workloads, strong consistency introduces significant latency overhead, especially on loosely-coupled volunteer nodes where propagation delays are high, often performing worse than the cloud. In contrast, eventual consistency on volunteer nodes offers a dramatic performance improvement, delivering latency comparable to that of dedicated edge nodes and significantly outperforming the cloud. This highlights a critical trade-off: for applications that can tolerate temporary data inconsistency, leveraging volunteer resources with an eventual consistency model provides a highly effective, low-latency storage solution at the edge.

These results highlight the importance of Cargo's application-aware design, which allows applications like face recognition, if tolerant of slight delays in updates, to choose eventual consistency and achieve significantly lower latency compared to a system enforcing strong consistency by default.

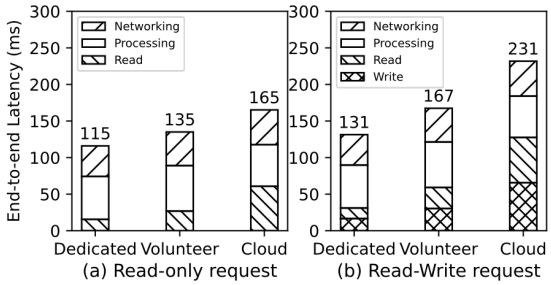


Figure 2.8: End-to-end latency of face recognition

End-to-end latency

Figure 2.8 provides a holistic view by breaking down the total end-to-end latency. The analysis reveals that while processing and initial network communication times remain relatively constant across configurations, the storage I/O (Read/Write) phase is the primary differentiator in performance. For instance, the low read latency of the dedicated edge directly translates to the best overall end-to-end performance. Conversely, the high write latency of a cloud backend significantly degrades the total application response time. This demonstrates that optimizing the storage layer, as achieved by Cargo's proximity-aware selection and tunable consistency, is critical for minimizing overall end-to-end latency for stateful edge applications.

2.5 Related Work

Storage at the edge can be categorized into offload (offload data to edge and sync with cloud), aggregate (Data collected from multiple devices to the edge) and P2P (data generated by one device shared with another) [29, 30]. Most of the existing storage systems focuses on offload and aggregate models. P2P storage is not explored much due to concerns of data security and synchronization difficulties across unreliable devices. CloudPath [31] uses PathStore [32], an eventually consistent datastore with persistent data on cloud and partial replicas on edge. The store may have a degraded performance when new data is queried frequently. SessionStore [33] is a hierarchical datastore that guarantees session consistency using session-aware reconciliation algorithms built on top of Cassandra [34] and hence support client mobility to an extend. DataFog [35] is

an IoT data management infrastructure which places replica based on spatial locality, addresses sudden surges in demand using a location-aware load balancing policy and evicts and compresses data based on temporal relevance. However, it does not support network proximity based node selection. FogStore [36] is a geo-distributed key-value infrastructure that places replicas based on latency of data access. Also, to ensure fault tolerance similar to DataFog, one of the replicas is kept at a remote location in FogStore. However, it does not take into account the limited storage capacities of heterogeneous storage nodes.

2.6 Conclusion

This chapter addressed the critical challenge of enabling stateful applications in dynamic, high-churn edge environments. We introduced Cargo, the in-situ storage layer for the Armada framework, designed to provide a reliable and low-latency persistence service by leveraging a mix of dedicated and volunteer edge resources. The design of Cargo decouples storage from volatile compute, providing stability and fault tolerance through a proximity-aware selection mechanism and automated data replication. Cargo also allows applications to specify key requirements, such as desired consistency levels, during deployment.

Our evaluation, conducted in both real-world and emulated environments, validated the effectiveness of this approach. The key findings demonstrate that:

1. Performance-aware selection is crucial: Cargo's selection strategy successfully identified the optimal storage node, outperforming naive geo-proximity by adapting to real-world network heterogeneity.

2. Volunteer resources are a viable fallback: While dedicated edge nodes offered the best performance (reducing end-to-end latency by up to 30% compared to the cloud), volunteer nodes provided a low-latency alternative (18% improvement over the cloud). This confirms that in the event of dedicated node failure, falling back to managed volunteer resources is significantly better than reverting to the high-latency cloud.

3 Tunable consistency is essential for the edge: Cargo's support for different consistency levels allows applications to make critical choices. For write-intensive workloads, an eventual consistency model on volunteer nodes delivered performance comparable to dedicated nodes, highlighting a critical latency-consistency trade-off that application-aware systems must support.

Ultimately, this work establishes that a decoupled, application-aware storage layer like Cargo is not just a feature but a fundamental requirement for building robust, stateful applications in heterogeneous and volatile edge-dense environments.

2.6.1 Chapter Takeaways

⇒ *Decoupled Storage is Essential for Stateful Edge Applications*: To ensure data persistence and application reliability in edge environments with high compute-node churn, the storage layer must be architecturally independent of the volatile compute layer.

⇒ *Client-Centric Performance Probing Outperforms Static Policies*: In heterogeneous edge networks, simple geo-proximity is an unreliable proxy for latency. A dynamic, performance-aware selection mechanism that probes from the client's (or compute node's) perspective is necessary to identify the truly lowest-latency storage node.

⇒ *Volunteer Resources are a Critical Component for Edge Resilience*: Managed volunteer nodes provide a powerful, low-latency fallback mechanism. Instead of defaulting to the high-latency cloud upon dedicated node failure, an edge framework should intelligently leverage available volunteer capacity to maintain service continuity and performance.

⇒ *Eventual Consistency is a Key Enabler for Performance*: For many write-intensive edge applications, the performance gains from using an eventual consistency model, especially on loosely-coupled volunteer resources, far outweigh the benefits of strong consistency, making it a critical, tunable option for developers.

Chapter 3

Location, Latency and Spatial awareness based Data placement Strategies at the Edge

3.1 Introduction

While the previous chapter demonstrated the benefits of edge storage under the assumption of pre-existing data replicas, this chapter addresses the antecedent and more fundamental challenges of data placement. Specifically, we explore methodologies for identifying optimal storage locations, determining the necessary number of replicas and dynamically managing their creation in response to application load.

IDC expects 55.9 billion connected devices by 2025 generating 79.4ZB data [37]. This data deluge is increasing the data gravity at the network's edge resulting in the emergence of edge applications [8]. Deploying application near to the source of data generation ensures low data access latency and low service latency with improved quality of service. For example, in case of AR/VR applications, the MTP(Motion to Photon) latency should be less than 20 ms for immersive experience [38]. Most of the data generated can be stored at the edge, utilized by edge applications and then be either sent to the cloud for persistent storage or be discarded. This temporary buffering strategy can decrease the traffic congestion on the wide area backhaul link and at the same

time give data owners more control to choose what data should be stored on the public cloud. However, there are multiple challenges one should address to ensure quality of service to users when it comes to storing data at the edge [5, 25]. The heterogeneity of edge storage nodes, limited elasticity, node churn, user mobility, time sensitive compute value of data and data privacy concerns present numerous research opportunities in edge storage. With the introduction of compliance laws like GDPR [39], it is also essential that data (example, health and trading) generated be persistently stored within a region.

The storage nodes at the edge are heterogeneous in terms of storage capacity, network bandwidth and request handling power. The limited elasticity and node churn makes it even more challenging. The placement of data under these environment conditions to provide best quality of service for application users by minimizing the average latency is not a trivial task. For easiness, we will call data generation source as Producers, data consuming applications as Consumers and edge storage servers as Hosts. The data generated by producers can be subscribed by multiple consumers, also a single consumer can subscribe to multiple producer data. Depending on an increase in demand by consumers and the available storage capacity, new replica storage servers may have to be created on the fly to continue service. Many of the existing data placement strategies take into consideration latency, geolocation, data popularity and spatial awareness along with system constraints to select a storage server for data placement. iFogStor [40] proposes an exact solution based on integer programming for data placement. However, to scale the solution, they proposed two heuristics with single replication to minimize the overall service latency. Having a single replica may not always be suitable for applications with a lot of consumers and can cause multiple read failures with increased latency. [41] extends the iFogStor strategy to allow multiple replicas across different edge nodes. We also place a similar multi-replica constraint in our problem formulation. Furthermore, works such as [18] projects servers based on geolocation and data popularity. However, this approach fails to consider network latency, which is not always proportional to physical distance in complex edge environments.

To address these limitations, this work proposes three key improvements that have a significant impact on the quality of service: ① Given the limited resources per edge server, we should consider the ingress and egress request handling capacity per server. If we go over the limit, then it will become a bottleneck leading to an increase in latency

(Producer/Consumer load constraint in Section). ② Identifying application specific characteristics like colocation of producers and consumers or density of producers and consumers at a location can help identify edge storage servers that will decrease the data transfer latency (Centroid-based replica selection in Section). ③ A combined use of geolocation, latency and spatial awareness can prune the search space for edge storage servers, resulting in less decision making time.

The main contributions in this paper are

- The proposal and analysis of two distinct system models for data placement: a store-and-forward model and a streaming model.
- An optimization formulation for the data placement problem, mapping it to the NP-Hard Generalized Assignment Problem.
- The design of a data placement framework and a comparative evaluation of three strategies operating within it: distance-based, latency-based, and a novel spatial-awareness-based approach.
- A comprehensive simulation-based evaluation demonstrating the performance trade-offs of the three strategies across both system models.

Section 3.3 details the system overview along with the optimization problem for data placement while considering multiple constraints. The spatial heuristic and additional data placement strategies are listed in Section 3.4. A comparison across the data placement strategies in showcased in Section 3.5. We conclude our work with a brief discussion of future work and the major findings in the paper in Section 3.7.

3.2 Motivation

The abstract Producer-Consumer-Host model can be mapped directly to a variety of emerging edge applications, each with distinct data flow patterns that motivate our two system models.

- **Autonomous Vehicles and AR Gaming (Streaming Model)**: In a vehicular network, a lead vehicle (a Producer) might stream real-time LiDAR or video data

to a Roadside Unit (a Host). This data is then immediately forwarded to trailing vehicles (Consumers) to enable cooperative maneuvering or hazard detection. Similarly, in an AR online game, a player's real-time actions and location (Producer) are streamed via an edge server (Host) to other nearby players (Consumers) to maintain a synchronized game state. In both cases, the critical performance metric is the end-to-end latency, as the value of the data diminishes rapidly with time.

- **Industrial IoT and Smart Farming (Store-and-Forward Model)**: On a factory floor, sensors on machinery (Producers) continuously generate data about machine health and operational status. This data is stored on an on-premise edge server (Host). Various applications, such as a predictive maintenance dashboard or a safety alert system (Consumers), can then query this stored data as needed. Here, the data has persistent value, and the key performance metric is the data retrieval latency for the consuming applications. The initial latency from the sensor to the server is less critical and can be amortized over many reads.

These scenarios highlight that a "one-size-fits-all" data placement strategy is insufficient. The optimal placement of data depends fundamentally on whether the application is latency-sensitive (streaming) or delay-tolerant (store-and-forward), motivating the distinct system models and strategies explored in this chapter.

3.3 System Overview

Figure 3.1, shows the system architecture that we consider. There are a set of data generators (producers), a set of edge storage nodes (hosts) and a set of application tasks (consumers) in the system. Producers generate data, store/stream it to a host, from where a consumer will collect the data. Gateways allow connection of producers to the internet and vice versa. A producer's data can be subscribed by multiple consumers and a consumer can subscribe to multiple producer data. The host nodes vary in storage capacity and ingress and egress request handling capacity. Host nodes with higher resource capacities are considered to have low latency to other nodes. A producer can directly communicate with a host once it is identified as a suitable location for data storage.

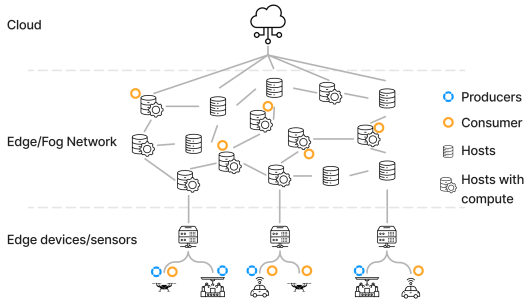


Figure 3.1: System architecture

3.3.1 System model

With this architecture, there can be two types of system models: Store-and-forward and Streaming.

- **Store-and-forward model**: In this model, the producer sends data to the host, where it is stored and then sent to any subscribed consumers. Once the data is available at the edge, any application can retrieve the data at any point of time. The edge applications that use the data are usually delay tolerant, for example, a car insurance company perusing through accident videos for validating insurance claims, or a city planning system that wants to rearrange the traffic over coming days based on the current day data. The quality of service here is determined by the data retrieval latency observed by the consumer. The latency of data transfer from the producer is amortized in this case.

- **Streaming model**: In this model, the producer will send the data to the host from where it is immediately transferred to the subscribed consumers. The data will be cached for a short period of time to account for any missing data during streaming. The edge applications in this case are latency-sensitive, for example, a augmented reality based robotic surgery handled by multiple surgeons requires the same data to be streamed to all involved doctors to avoid any life threatening

situations. The quality of service here is determined by end-to-end latency observed by the consumer. The data transfer latency from producer to host should be considered.

3.3.2 Problem Formulation

Given sets of Producers (P), Hosts (H) and Consumers (C) with sizes p, h and c respectively. Each producer p_i sends data of size $datasize_{p_i}$ to atmost r hosts to meet the demands of subscribed consumers. The value of r may vary from producer to producer, however, here we take $r = replica_threshold$ for all producers. A consumer (c_k) can subscribe to more than one producer. For simplicity, we assumed the data transfer unit is b bytes. Each host has a storage capacity of cap_{h_j}, a producer load threshold $pload_{h_j}$ (number of concurrent producer connections) and a consumer load threshold $cload_{h_j}$. Each consumer maintains a binary subscription list $csub$ of size $(1 \times p)$, where, $csub_{ki} = 1$, if c_k subscribes to p_i, otherwise $csub_{ki} = 0$. The binary matrix, phc of size $(p \times h \times c)$ is used to represent the paths from a producer to a consumer via a host. If $phc_{ijk} = 1$, there exists a path from producer (p_i) to consumer (c_k) via host (h_j), otherwise $phc_{ijk} = 0$.

- *Load constraint*: A producer p_i or a consumer c_j can use a host h_j only if the addition of new connection is within the producer or consumer load threshold respectively. If plt_{h_j} is the producer load threshold and clt_{h_j} is the consumer load threshold for host h_j, then

$$\sum_{i=1}^{p}(\sum_{k=1}^{c} phc_{ijk} > 0) \leq plt_{h_j} \tag{1}$$

$$\sum_{k=1}^{c}(\sum_{i=1}^{p} phc_{ijk} > 0) \leq clt_{h_j} \tag{2}$$

- *Storage constraint*: A host (h_j) can store data from a producer (p_i) only if $datasize_{p_i}$ is less than the available storage capacity cap_{h_j} of the host.

$$\sum_{i=1}^{p} datasize_{p_i} * (\sum_{k=1}^{c} phc_{ijk} > 0) \leq cap_{h_j} \tag{3}$$

- *Single path constraint*: There exist a single path from a producer p_i to a consumer c_k via a host h_j, provided c_k is subscribed to p_i, i.e., $csub_{ki} = 1$.

$$\sum_{j=1}^{h}(phc_{ijk} * csub_{ki}) = 1 \tag{4}$$

- *Replica count constraint*: The maximum number of replicas alloted to a producer (p_i) is bounded by a replica threshold. For simplicity, we are taking the same threshold rt across all producers.

$$\sum_{j=1}^{h}(\sum_{k=1}^{c} phc_{ijk} > 0) \leq rt \tag{5}$$

Given the above constraints, we need to ensure that selected path (phc_{ijk}) for data transfer from producer (p_i) to consumer (c_k) via host (h_j) has a low latency. If the latency of transferring b units of data between p_i and h_j is lat_{ij} and that between h_j and c_k is lat_{jk}, then the total latency is given by

$$lat\_phc_{ijk} = (lat_{ij} + lat_{jk}) * \frac{datasize_{p_i}}{b} * phc_{ijk} \tag{6}$$

For store-and-forward system model, the effect of lat_{ij} will be amortized.

Objective

The objective of our data placement algorithm is to find the hosts where data from the producers can be stored for consumption by consumers, while providing minimum average latency. This means filling the binary matrix phc by minimizing (6) for all

producers, hosts and consumers.

$$
\begin{aligned}
Minimize \quad & \frac{\sum_{i=1}^{p}\sum_{j=1}^{h}\sum_{k=1}^{c} lat\_phc_{ijk}}{\sum_{i=1}^{p}\sum_{j=1}^{h}\sum_{k=1}^{c} phc_{ijk}} \\
subject\ to \quad & \sum_{i=1}^{p}(\sum_{k=1}^{c} phc_{ijk} > 0) \leq plt_{h_j} \\
& \sum_{k=1}^{c}(\sum_{i=1}^{p} phc_{ijk} > 0) \leq clt_{h_j} \\
& \sum_{i=1}^{p} datasize_{p_i} * (\sum_{k=1}^{c} phc_{ijk} > 0) \leq cap_{h_j} \\
& \sum_{j=1}^{h}(phc_{ijk} * csub_{ki}) = 1 \\
& \sum_{j=1}^{h}(\sum_{k=1}^{c} phc_{ijk} > 0) \leq rt
\end{aligned}
\tag{7}
$$

3.3.3 Optimization Solution

The objective (7) can be represented as a Mixed-Integer Non-Linear Programming (MINLP) problem. IBM CPLEX Solver is used to model and solve the placement problem. As it was mentioned, being an NP-Hard problem, as the number of actors in the system increases the execution time also increases which is not suitable for latency-sensitive applications. Hence, we propose a data placement framework that uses three strategies to scale the solution in the next section.

3.4 Data Placement Strategies

In section 3.3, the system model and the MINLP model were discussed. Being NP-Hard [40], we cannot solve the optimization problem in polynomial time as the number of host nodes increase. We present three strategies which take into consideration all the required factor discussed in section 3.3 to provide a approximate solution.

1. **Distance based selection**: The selection of nearest edge servers to the source

of data generation allows to ensure data is stored within a specific locality. Applications where it is known that both producers and consumers are co-located will benefit from distance based selection.

2. **Latency based selection**: Distance based selection may not always provide you with lowest latency route to an edge server. In this strategy, the selection of edge servers is based on the latency of a producer to an edge server or host. The host that can provide satisfactory latency (within latency threshold) is selected for data storage.

3. **Spaital heuristic based selection**: In a dense edge environment with thousands of edge servers, using distance and latency based selection may incur higher decision making latency. This overhead can be mitigated by effectively pruning the search space.. In this strategy, we prune the search space associated with a producer using a spatial data structure (RTree) that groups hosts based on geo-locality. Once a specific group is identified, the best host with low latency to producer is selected for data storage.

Before discussing the details of the algorithm, we first explore the matchmaker (Fig. 3.2), which is the decision making element associated with a group of producers, consumers and hosts. While the data transfer is decentralized (producer-to-host, host-to-consumer), a centralized matchmaker is employed to maintain a global view of resource availability (load, capacity) and network conditions, enabling more optimal and globally-aware placement decisions than a purely local approach would allow. It resides on one of the edge servers which may have storage. The matchmaker has two main modules: Network and replica. The Network module sends periodic requests to producers and consumers to fetch the dynamically changing network latency and geo-location information. The Replica module decides how many replicas are required for producer data, depending on the demand and availability of resources.

3.4.1 System Workflow

The system workflow, illustrated in Figure 3.3, follows a registration and discovery pattern. First, Producers register with the matchmaker, providing their location and

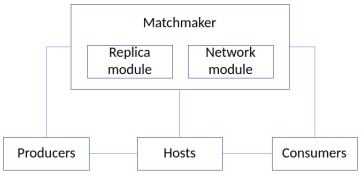


Figure 3.2: Matchmaker, the deicison making component

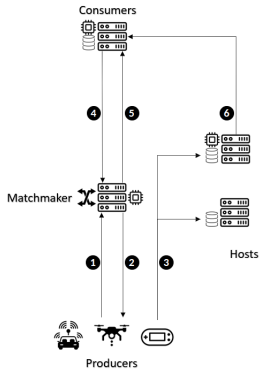


Figure 3.3: System workflow

data requirements ①. The matchmaker then identifies and returns a list of suitable Hosts ②. The Producer establishes a direct connection to a Host to begin storing its data ③. Subsequently, a Consumer registers, providing its own location and a list of Producer data it wishes to subscribe to ⑤. The matchmaker cross-references this subscription with its record of Host locations and provides the Consumer with the optimal Host to connect to for data retrieval ⑥.

The matchmaker runs one of the above strategies in *HostSelect* (algorithm 1) in two cases. ⓐ Producer enters the system for the first time and ⓑ Dynamic replication caused by exceed load on a host node.

For distance and latency based strategies (algorithm 2), the matchmaker first orders host nodes based on their distance or latency (line 3). The best *nreplica* nodes are then selected by taking into account the storage capacity and load (line 4). As multiple decisions are taken at the same time, (line 4) will be executed in a loop until the required number of replicas are found. In cases where all the resources are exhausted, the producers are denied service and will have to wait for a new host to join or existing hosts to become free. The information of the selected hosts are then shared back to the

Algorithm 1 Host node selection for Producer

```
1: procedure ProducerHostSelect(pinfo, nreplica)
2:     {id, loc, sz} ← {pinfo.id, pinfo.loc, pinfo.datasize}
3:     pinfo.hinfos ← HostSelect(id, loc, sz, nreplica)
4:     return pinfo.hinfos
```

Algorithm 2 Distance/Latency based

```
1: procedure HostSelect(id, loc, sz, nreplica)
2:     nodes ← GetHostNodes()
3:     ordered_nodes ← SortByDistanceOrLatency(id, loc, nodes)
4:     host_nodes ← SelectViableNodes(sz, nreplica)
5:     return host_nodes
```

producer to start the data transfer. The time complexity associated with host selection is $O(nlogn)$, where n is the number of host nodes under consideration.

The *SelectViableNodes* function iterates through the sorted list of potential hosts, selecting the first *nreplica* nodes that satisfy the producer's data size requirement (sz) and have not exceeded their current load and storage capacity thresholds.

As for the spatial heuristic (algorithm 3), the matchmaker identifies potential host nodes within the minimum bounding rectangle (MBR) of the producer based on latency, storage capacity and load (line 2) (Fig. 3.4.(a)). There is a chance that within the MBR of the producer, all the host nodes are full. In such a case we extend the search to the next MBR which is having the least MinDist [42]. MinDist is the least distance of a point to any of the nearby MBRs. In most of our experiments, we were able to see that, the required host node will be found by this step (line 4) (Fig. 3.4.(b)). However, if that is not the case, then we extend the search outwards in a concentric circle manner to search for a potential host node, in an incremental fashion (line 6) (Fig. 3.4.(c)). Once the potential hosts are identified, they are then ordered based on increasing order of latency to producers and the top *nreplica* nodes with sufficient storage and load are selected and returned to the producer (line 9). If we are not able to find host nodes, then the entire procedure is repeated taking into account the churn and load change of host nodes. The matchmaker stores the replica information. The time complexity for spatial heuristic host selection is $O(log_M(n))$, where M is the maximum number of children per node in RTree and n is the number of host nodes under consideration.

Algorithm 3 Spatial Heuristic

```
1: procedure HostSelect(id, loc, sz, nreplica)
2:     nodes ← SD.GetHostNodes(id, loc, sz)          ▷ SD is the RTree structure
3:     if nodes.empty() then
4:         nodes ← SD.SearchMinDist(id, loc, sz)
5:         while nodes.empty() do
6:             nodes ← SD.ConcentricSearch(id, loc, sz)
7:     host_nodes ← SortByLatencyAndSelectViableNodes(nodes, nreplica)
8:     return host_nodes
```

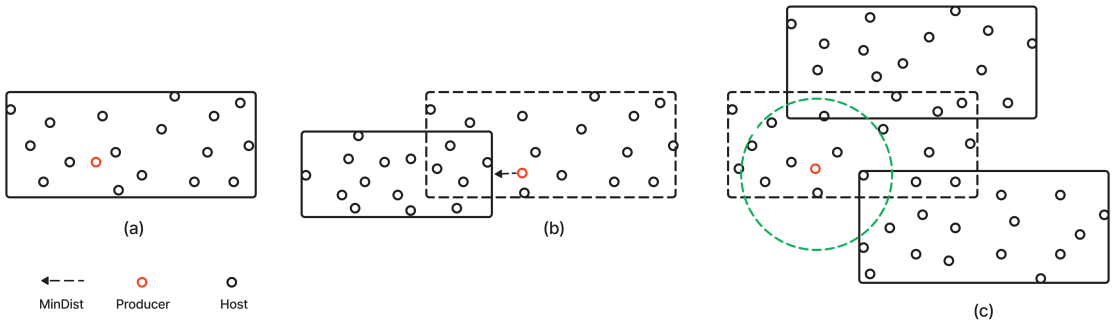


Figure 3.4: Host search (a) Search in MBR of Producer, (b) Search in MinDist MBR, (c) Search in Concentric Circles with producer as center

An RTree is chosen for this purpose due to its efficiency in indexing multi-dimensional spatial data, allowing for fast nearest-neighbor and range queries, which are essential for our geo-locality-based pruning.

The consumer is alloted a host to connect to depending on the subscription list provided at the registration. Algorithm 4 shows the pseudocode for host selection process from the consumer side. Based on the subscription list provided, the matchmaker will identify all the replicas for each producer. Then based on latency and load, a replica is allocated to the consumer (line 4-5). There is a chance that all the existing replicas are already alloted to other consumers. This can lead to dynamic replication by calling *HostSelect* with *nreplica=1* (lines 6-11). Once a host is identified and the data transfer from the producer starts, the information is shared back to the consumer (lines 13).

Algorithm 4 Host node selection for Consumer

```
1:  procedure ConsumerHostSelect(prods, subinfo)
2:      hinfo ← {}
3:      for prod_id ∈ subinfo do
4:          prod ← prods[prod_id]
5:          hinfo ← GetSubHost(prod)          ▷ Sort by latency and then select by load
6:          while hinfo.empty() do
7:              {id, loc, sz} ← {prod.id, prod.loc, prod.datasize}
8:              new_replica ← HostSelect(id, loc, sz, 1)
9:              prod.hinfo.append(new_replica)
10:             hinfo.append(new_replica)
11:     return hinfo
```

3.5 Evaluation

3.5.1 Experimental Setup

The simulation experiments are run on a Linux machine with 64GB RAM and 24 cores for the simulation. We select 5ms-10ms, 10ms-15ms, and 15ms-20ms ranges for high-capacity storage, medium-capacity storage, and low-capacity storage host nodes. The location associated with producers, hosts, and consumers was taken from the Social IoT real-time dataset [43]. The storage capacity of host nodes ranges from 32GB to 1TB, generally correlated with network quality (i.e., lower latency nodes typically have higher capacity). Each host has defined load thresholds: the producer load limit ranges from 15 to 25 concurrent connections, and the consumer load limit ranges from 25 to 55 connections, generally scaling with the node's capacity. Producers can generate data in the range of 1GB-32GB. The consumer arrival follows Poisson distribution with a mean inter-arrival time of 5ms. Each producer will send chunks of size 1024 bytes to hosts until it reaches the data size to be generated. The RTree parameters M (maximum number of children within a node) and m (minimum number of children within a node) are set to 40 and 20, respectively.

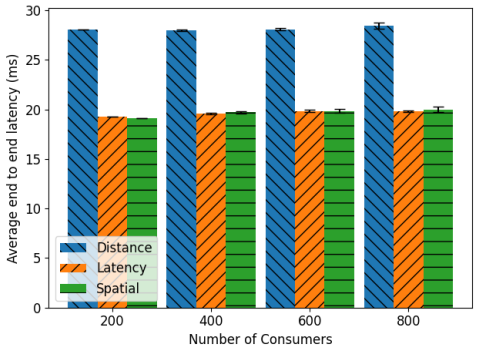


Figure 3.5: Average end-to-end latency. The number of hosts and producers is set to 50 and 100, respectively. Distance-based looks only at location information, leading to the selection of nodes with high latency. In contrast, latency-based and spatial-based consider latency resulting in low end-to-end latency.

3.5.2 Simulation experiments

End-to-end latency

End-to-end latency provides a measure of the quality of service. In this experiment, we simulate 50 hosts, 100 producers, and a varying number of consumers (200-800). It can be seen in Figure. 3.5, the average end-to-end latency remains almost the same across all the difference (host, producer, consumer) configurations. This is because the number of times dynamic replication occurs is much less than the number of chunks transferred. A detailed look at the replica overhead is shown in section 3.5.2. The distance-based strategy takes more time as it does not consider the selected host's latency. There is also the chance that the host node has less resource capacity, leading to more replications. In contrast, both latency-based and spatial-based strategies prioritize low-latency connections. Latency-based selects the host with the globally lowest latency meeting constraints, while spatial-based selects the lowest latency host within its pruned search space. In this sparse environment (50 hosts), both approaches effectively identify high-quality hosts, leading to similar average end-to-end latency. We will look at a scenario where spatial-based will outperform latency-based in section 3.5.2.

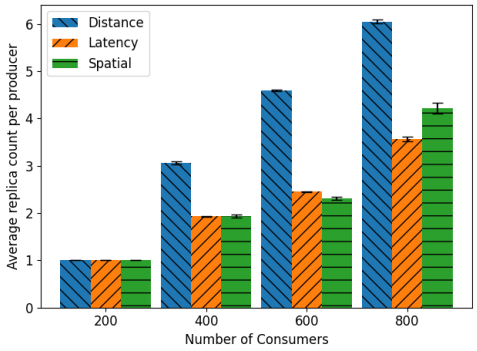


Figure 3.6: Average replica count per producer. The number of hosts and producers is set to 50 and 100 respectively. In the given environmental setting, distance-based may end up selecting hosts with less resource capacity. This selection leads to the creation of more replicas.

Average replica count per producer

For the same simulation scenario discussed above, the average count of replicas per producer is shown in Figure 3.6. It can be seen that latency-based and spatial-based outperform distance-based in all the configurations. Distance-based does not consider host load or capacity during initial selection, making it more likely to choose nodes that quickly become saturated, thus triggering more frequent dynamic replications. Spatial-based shows a similar replica count to latency-based. However, there is a chance that spatial selects host nodes with comparatively fewer resources. This selection leads to more replicas, as shown in the configuration (50,100,800).

Average replication overhead per consumer

The average replication overhead, defined as the time from a consumer's request for a new replica until it receives the first data chunk, surprisingly shows little variation across the strategies in this sparse setting (Figure 3.7). This includes Matchmaker decision time, producer-host connection setup, and initial data transfer. Figure 3.8, showing the distribution of overheads, provides the insight. The distance-based strategy exhibits a wide distribution, reflecting its tendency to select both high-latency (leading to slow setup/transfer) and occasionally low-latency hosts. In contrast, latency-based and spatial-based strategies initially select low-latency hosts, resulting in faster overheads

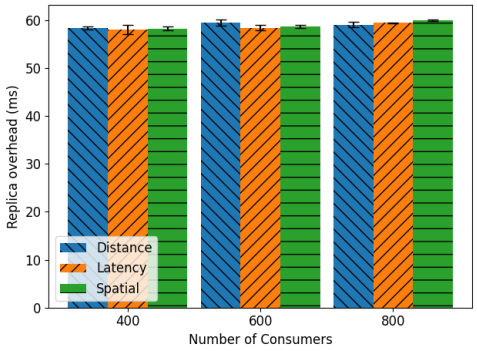


Figure 3.7: Average replication overhead per consumer. The number of hosts and producers is set to 50 and 100, respectively. Given the low host count, and the high number of chunks transferred, the average overhead across all the strategies is observed to be the same.

(peaks at lower values). However, as these optimal hosts become saturated, subsequent replications are forced onto less ideal hosts, increasing their overhead and shifting parts of their distribution rightward. In this specific configuration (50 hosts), the averaging effect across these different distributions leads to comparable mean overhead values.

Average consumer replica selection time

This experiment isolates the average replica selection time at the Matchmaker, simulating a dense environment with 5000 hosts. As shown in Figure 3.9, the spatial-based strategy exhibits significantly lower decision times compared to both latency-based and distance-based approaches. This directly demonstrates the benefit of the R-Tree based search space pruning, which avoids the $O(n \log n)$ sorting required by the other methods. While this Matchmaker overhead component was not the dominant factor in the overall replication overhead in the sparse (50-host) environment, it becomes a critical performance bottleneck in denser deployments, highlighting the scalability advantage of the spatial heuristic.

The simulation results conclusively show that application-aware strategies (latency-based and spatial-based) significantly outperform the naive distance-based approach in terms of end-to-end latency and replica efficiency. While latency-based offers near-optimal host selection, its decision-making overhead scales poorly. The spatial-based

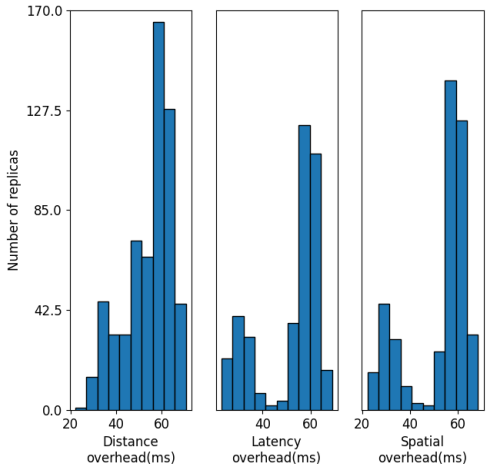


Figure 3.8: Replica overhead distribution. The replica overhead is distributed across all bins in distance-based, leading to similar overhead as latency-based and spatial-based.

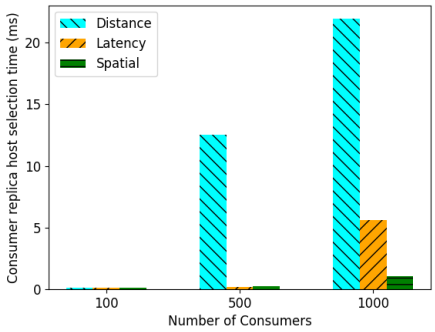


Figure 3.9: Average consumer replica selection time overhead. The number of hosts and producers is set to 5000 and 50, respectively. The replica selection overhead increases at the Matchmaker with the number of consumers concurrently requesting replicas. Spatial-based can focus on the pruned search space resulting in less time.

strategy emerges as the most scalable solution, effectively balancing near-optimal host selection with significantly lower decision-making time due to its search space pruning, making it particularly well-suited for dense edge environments.

3.6 Related Work

Over the past five years, multiple approaches have been proposed for data placement at the edge. iFogStor [40] models the data placement as GAP [44], an NP-Hard problem. Its main goal is to identify a single replica storage node where data from the producer be kept to minimize the overall latency between producers and consumers. As the solution cannot scale, a heuristic based on geographical zoning is proposed. The single replication of data may not always be suitable as there can be increased requests resulting in network and storage throttling. Also, if there is an inter-regional flow of data, the zone-specific solution may be sub-optimal. To solve this problem, iFogStorG [45] proposed a divide and conquer approach. It divided the entire edge infrastructure into several disconnected and balanced parts to ensure minimized data flow across parts. Within each identified part, the iFogStor approach was run to get the local decision and then combined to get suitable global placement. Here, only a single replica is associated with a producer making it unsuitable for high request scenarios. To resolve the single replica issue, iFogStorM [41] was proposed. The model adds a constraint to have more than one replica per producer. Similar to iFogStor, iFogStorM cannot be solved in polynomial time. Hence the authors proposed the MultiCopyStorage heuristic, which greedily allows a consumer to select the low latency node among the replicas. It also curbs the replica count when increasing the count does not significantly impact the overall latency. One issue with this technique is that the number of replicas may be too high in an already restricted edge environment. Also, focussing on replicas too far away from consumer may not be required as edge application users are mostly colocated (autonomous vehicles, AR/VR games). [46] introduces iFogStorS for small infrastructures that uses shortest path between producers and consumers; and iFogStorP for large infrastructures that uses P-median [47] to place P replicas. Compared to MultiCopyStorage which in parallel sends updates from producer to replicas, iFogStorS/P sends data to one replica, which in turn updates others.

Scientific workflows usually have very large generated datasets stored across multiple cloud data centers leading to high transmission delay. [48] proposes a genetic, self-adaptive, discrete particle swarm optimization data placement strategy (GA-DPSO) that utilizes both the cloud and the edge. The approach does not consider the highly heterogeneous feature of edge nodes. [49] takes into consideration the storage capacity at each edge site and the data transmission cost across cloud nodes to make a placement decision. They propose a discrete particle swarm optimization with differential evolution to identify the locations where shared data across multiple scientific workflows can be placed to minimize the transmission time.

In [50], the switches and data indices of unstructured data are associated with coordinates in a virtual space. The data index is stored on servers that is connected to a switch which is closest in the virtual space. In [51], inspired by [50], the data is placed at the center of a dense network in a virtual space to ensure shorter distance to all the areas in a region followed by popularity based replica placement. [52] jointly places tasks and data, where each block of data is assigned a popularity value to help make decision on data placement.

FogStore [36], a key-value store, places a set of replicas within the vicinity of the clients and another set of replicas away from the clients to ensure fault-tolerance. It provides differential consistency for data depending on situation-awareness of applciations. DataFog [35] is a data management platform for IoT which uses spatial proximity to identify the location of replicas. Similar to FogStore, it also keep a few replicas in remote locations for fault tolerance. EdgeKV [53] is a decentralized storage system for general purpose tasks with fault-tolerance, reliability guarantees and strong consistency. Distributed Hash tables are used in EdgeKV to identify locations to store data across different edge nodes.

Data placement is a well researched topic in distributed systems [54–63]. They mainly focus on reducing the network latency, reducing proximity of dedicated servers to clients, data popularity, adapting to dynamic workloads, partitioning of data to adapt to server sizes and dynamically configuring replicas based on application requirements. Many of the existing distributed databases [34, 64, 65] use consistent hashing to store data across different nodes in a load balanced manner.

3.7 Conclusion

This chapter addressed the critical problem of efficient data placement in heterogeneous edge environments, aiming to minimize latency for data-intensive applications. Recognizing the limitations of centralized cloud storage and the unique constraints of the edge, we first defined two distinct operational models, Store-and-forward and Streaming, reflecting different application needs. We then formulated the data placement challenge as a MINLP optimization problem, explicitly incorporating crucial factors like server location, network latency, load thresholds (producer and consumer), and storage capacity.

Acknowledging the NP-Hard nature of the optimal solution , we designed and evaluated a data placement framework featuring three heuristic strategies: distance-based, latency-based, and a novel spatial-awareness-based approach using R-Tree pruning. Our simulation results yielded clear insights into their performance trade-offs:

1. Both latency-based and spatial-based strategies significantly outperformed the naive distance-based approach in terms of end-to-end latency and replica efficiency, demonstrating the necessity of considering network performance and resource capacity.

2. In sparse edge environments, latency-based and spatial-based strategies showed comparable performance, effectively identifying low-latency hosts.

3. However, in simulated dense edge environments, the spatial-based strategy proved significantly more scalable. Its R-Tree pruning mechanism drastically reduced the Matchmaker's decision-making time compared to the latency-based approach, offering a crucial advantage when handling numerous concurrent requests.

This work underscores the importance of application and environment-aware data placement. While latency-based selection provides near-optimal results in sparser settings, the spatial heuristic offers a compelling balance of performance and scalability, particularly for future dense edge deployments.

3.7.1 Chapter Takeaways

⇒ *Latency Matters More Than Distance*: Simple distance-based data placement is often suboptimal in edge environments due to network heterogeneity; latency-aware strategies yield significantly better application performance.

⇒ *Load Constraints are Crucial*: Data placement decisions must account for server load (ingress/egress) and storage capacity to avoid bottlenecks and minimize excessive dynamic replication.

⇒ *Scalability Requires Search Space Pruning*: While selecting the absolute lowest-latency server (Latency-based) is effective, the decision-making overhead becomes prohibitive in dense environments. Spatial heuristics offer a scalable alternative by efficiently pruning the search space.

⇒ *Context Dictates Strategy*: The optimal data placement strategy depends on the edge environment's density. Latency-based may suffice for sparse deployments, while spatial-awareness is critical for scalability in dense scenarios.

Chapter 4

ASTRA: Associaton, Spatial proximity and Temporal Relevance based Adaptive prefetching for Edge AR

4.1 Introduction

Mobile augmented reality (MAR) is rapidly evolving pervasive tool with applications spanning diverse sectors such as gaming, entertainment, education, industry and healthcare. AR enhances user perception by overlaying digital content onto the real world, creating immersive experiences. However, delivering quality AR experiences presents unique challenges, mainly due to the stringent demands of high bandwidth, ultra-low latency and real-time responsiveness. These requirements make existing cloud infrastructures unsuitable for timely AR service delivery, necessitating the need for innovative solutions.

Edge computing has emerged as a promising computing paradigm to address these challenges by bringing computation and storage resources closer to the end-user at the network edge. The reduced distance, increase of bandwidth and high speed infrastructures such as 5G(6G), significantly lowers the latency and improves the performance

of MAR applications. Complementing edge computing, prefetching techniques anticipate users needs and proactively fetch AR artifacts to the edge/end-user device from the cloud. This proactiveness further mitigates latency, enhances user experience and optimizes edge resource utilization.

Several recent research efforts focus on leveraging the predictability of user movement within both virtual and physical environments to enhance AR content delivery. By anticipating where a user is likely to look or move next, prefetching strategies [66–68] proactively retrieve relevant AR artifacts onto edge servers, thereby reducing interaction latency. While anticipating user movement via viewport [69] or Field of View (FoV) [70] or gaze [71] offers a valuable approach, these strategies face challenges under significant edge server resource constraints. Given that edge servers operate under significant resource constraints, prefetching strategies must incorporate mechanisms to avoid server overload. For instance, within a predicted viewport/FoV, fetching all potentially visible AR artifacts may be unnecessary and inefficient, as users typically interact with only a subset of available content. *This highlights the challenge and opportunity in determining precisely which artifacts possess sufficient relevance to warrant prefetching under these constraints.*

Beyond user movement, AR prefetching strategies can be enhanced by incorporating a broader range of contextual cues. Such comprehensive contextual awareness extends beyond tracking where the user is looking or moving, rather the wider circumstances of their interaction. The relevant factors include the specific task the user is performing, their history of past interactions, and potentially the dynamics within collaborative AR scenarios [70, 72]. Leveraging these richer contextual elements allows prefetching decisions to become more accurately tailored to the user's anticipated needs. *Ultimately, combining this detailed contextual understanding with AR artifact relevance enables selective content prioritization, a crucial capability for making AR applications feasible under strict edge resource constraints.*

This paper investigates optimizing edge prefetching in AR applications by analyzing user-object interaction patterns and AR-specific properties. Given the context-dependent nature of AR interactions, we argue that effective prefetching strategies should prioritize objects likely to enter the user's field of view (FoV). We hypothesize that predictable patterns in user access behavior can be leveraged to anticipate

future interactions and prefetch relevant virtual objects at the edge, thereby improving rendering performance and enhancing the overall AR experience.

We present ASTRA, a novel prefetching framework tailored for mobile augmented reality edge caches by integrating object associations derived from user interaction patterns with spatial awareness based on the user's physical location and field of view . This approach goes beyond traditional prefetching strategies by employing a *association factor* that considers both the frequency and recency of object co-access, alongside a *lazy fetching strategy* that prioritizes prefetching only when the user is in close proximity to the virtual objects . Furthermore, ASTRA incorporates an adaptive tuning algorithm for minimum support in association rule generation to minimize the computation overhead, making it a distinct and effective solution for enhancing user experience in AR applications by ensuring timely virtual object availability.

The novel contribution in this paper are:

- ASTRA, an edge prefetching framework for MAR that prioritizes object prefetching from cloud-to-edge based on Field of View (FoV), contextual information, temporal relevance and spatial proximity.
- An efficient prefetching mechanism combining association rule mining and spatial awareness for low overhead decisions, coupled with a proximity triggered lazy fetching approach to conserve edge resources.
- An adaptive minimum support tuning mechanism to maintain optimal performance over time by adjusting to evolving user behavior.
- Comprehensive simulation results, using both synthetic and real-world traces, demonstrating that ASTRA significantly improves hit rates (up to 35%) and reduces end-to-end latency (up to 11%) compared to baseline strategies, with adaptive tuning providing further hit rate gains (up to 10%).

The subsequent sections of this paper will delve into a detailed explanation of the motivation, the ASTRA framework, a comprehensive presentation and discussion of the results obtained, and finally, a conclusion outlining the key findings and potential directions for future research.

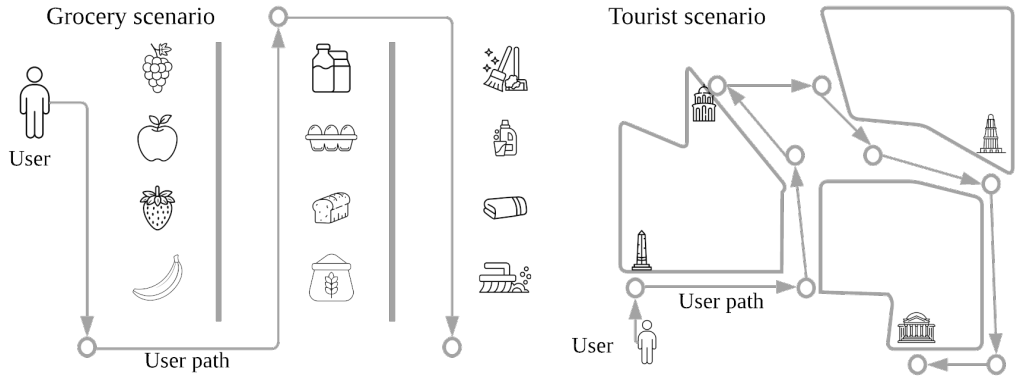


Figure 4.1: Retail and Tourism application scenarios

4.2 Motivation

Mobile augmented reality is increasingly deployed across diverse domains to improve user experience, from cultural heritage exploration and industrial operations to everyday retail engagement, by seamlessly blending digital information with the physical world. For MAR it is essential to provide contextually relevant information precisely when and where it is needed and it is contingent upon real-time delivery of digital assets. However, the significant latency in retrieving and rendering this content, can severely degrade user immersion and task efficiency, presenting a major obstacle to widespread adoption. While edge computing architectures aim to reduce network delays by locating resources closer to the user, achieving the perception of instantaneous interaction often demands proactive strategies. Prefetching, the anticipatory loading of required data, emerges as a critical technique. Yet, the core challenge persists: *how to accurately predict future user needs to make intelligent prefetching decisions, especially given the resource limitations of edge servers?* This section explores motivating application scenarios (Figure 4.1) where optimizing prefetching is paramount for realizing the full potential of AR, highlighting the specific need for context-aware strategies.

- `Museums & Tourism`: In scenarios like museums or navigation, user movement is

inherently spatial. The historical data could be used to derive associations between points of interest. By prefetching AR content (like audio guides, historical context, or navigation arrows) only for associated exhibits or landmarks that are also physically nearby, we prioritize the most probable next interaction. This targeted approach ensures the limited edge storage is used for immediately relevant assets, enhancing the user's exploration smoothly without wasting space on associated items they aren't close enough to view yet.

- `Manufacturing & Maintenance`: During industrial tasks, procedures follow a sequence of interactions (associations), frequently involving components located near each other. Prefetching AR instructions, diagrams, or safety warnings requires careful resource management. Focusing prefetching efforts on steps/components that are both historically the next logical action and physically proximate to the technician's current focus makes optimal use of constrained edge storage. It delivers the most crucial, immediately needed AR guidance without caching data for steps further down the line or unrelated nearby equipment.

- `Retail & Home Improvement`: When using AR for shopping or design, users often evaluate items in relation to their physical surroundings or nearby complementary products. The historical data reveals strong associations between items (e.g., sofas and coffee tables, dresses and specific accessories). Combining this with proximity drastically narrows down the possible number of AR objects to prefetch. This ensures the edge cache holds high-probability, actionable options pertinent to the user's immediate context.

Across these diverse application scenarios, the common characteristics underscore the need for intelligent AR prefetching. User interaction is fundamentally context-dependent, often following predictable sequences driven by spatial layouts (Museums), procedural steps (Manufacturing), or logical relationships between objects (Retail). Simultaneously, the effectiveness of AR hinges on low latency delivery of relevant information, a challenge given the potentially vast amount of available digital content and the constrained resources of edge servers. Simply prefetching everything associated is infeasible. Therefore, a strategy that not only leverages historical associations to predict user intent but also intelligently filters to fetch only the most relevant associated objects

is crucial for efficiency. Furthermore, given that AR inherently blends the digital with the physical, incorporating user proximity as a key factor ensures that the limited edge cache prioritizes assets for objects within the user's immediate vicinity. This combined approach, leveraging historical associations filtered by spatial proximity and temporal relevance, core components of the ASTRA framework presented in this chapter, is essential to optimize edge resource utilization and deliver the responsive, seamless experience users expect from AR applications.

4.3 ASTRA Prefetching Framework

4.3.1 System Architecture Overview

This section proposes the ASTRA prefetching system, designed for application scenarios mentioned in Section 4.2. The primary goal is to minimize latency when displaying AR content by proactively caching relevant assets on a nearby edge server. Our approach combines historical item interaction associations with real-time user proximity. It also considers the relevance of object associations and their recency to enrich the contextual information to optimize prefetching decisions. The architecture follows a Client-Edge-Cloud model as shown in Figure 4.2.

The main components in MAR architecture considered for ASTRA prefetching are

- AR Client: Runs on the user's mobile device or AR Head mounted devices. It continuously captures video frames, estimates device's 6DoF pose (SLAM), tracks identified objects position relative to the device's pose and renders 3D assets. The AR client transfers compressed keyframes, objects of interest (any interacted item) and location/pose. We consider binary glTF (glb) [73] 3D file formats of virtual objects which consists of the 3D model's geometry, textures and animations.

- Edge Server: It does object detection and recognition, estimates initial pose for newly identified objects, hosts the Prefetching Engine, manages the limited edge cache, fetches virtual objects from cloud and handles communication with clients.

- Cloud Backend: Stores the complete AR Asset Repository and the Association Engine, which processes historical data offline/periodically. Using algorithms like

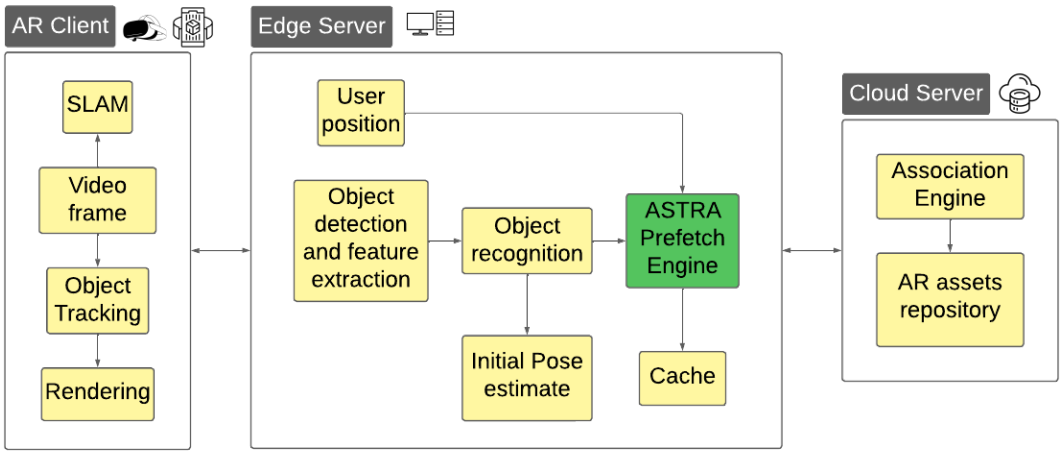


Figure 4.2: MAR architecture

Apriori or FP-Growth, it generates association rules. These rules quantify the likelihood of items being purchased or viewed together. The generated ruleset is periodically pushed to Edge Server's Prefetching Engine.

4.3.2 ASTRA Prefetching Engine

The Prefetching Engine runs on the Edge Server, leveraging association rulesets and user context to proactively cache AR assets. When the AR client sends keyframes, objects of interest and user location, the server first attempts recognition and pose estimation. The server checks the cache to see Aif R assets corresponding to objects of interests are available. If there is a miss, the engine uses association rules and the current session's interaction history to identify potentially relevant associated objects. These candidates are filtered first by an *association factor* (measuring association recency/relevance) and then by *proximity* (measuring physical closeness to the user). Let $a_t(o)$ represent the association count of AR asset o at time t. Let $\psi_t(o)$ the association factor of o at time t. Let λ be the smoothing value. It is represented by $2/(1 + window)$ where $window$ is the

window frame of previous object interactions which are relevant. Then,

$$\psi_t(o) = (\lambda \cdot a_t(o)) + ((1 - \lambda) \cdot \psi_{t-1}(o)) \tag{4.1}$$

The *window* value could be modified according to the amount of contextual information required.

Assets for the original objects of interest (if not already cached) and for associated objects passing both score thresholds are then fetched from the cloud and stored in the edge cache. Assets for the original objects of interest requested by the client are fetched immediately (if not cached), while assets for associated objects passing the filters are prefetched in the background to avoid delaying the primary response. Associated objects that are contextually relevant but currently too distant (failing the proximity check) are placed in a lazy prefetch buffer. These may be fetched later if the user moves closer to their location while they remain relevant (high association factor).

Finally, the edge server returns the poses of recognized objects and the assets for the requested objects of interest (from cache or just fetched) to the AR client. The proactively cached assets for associated items ensure faster loading upon subsequent encounters.

4.3.3 Minimum support tuning

The association rules are currently generated on the Cloud server and periodically distributed to the Prefetching Engine on the Edge. This centralized, periodic approach struggles to adapt to dynamic user workloads, risking the use of stale rules and increased cache miss rates between updates. To enhance responsiveness and adaptability, we propose relocating rule generation to execute directly on the Edge Server alongside the Prefetching Engine. Running as a background process to minimize interference with primary tasks, this allows rule generation to leverage local user interaction patterns for more timely and relevant prefetching decisions. The process of generating new association rules, which incorporates an adaptive minimum support tuning heuristic, is initiated by the Prefetching engine under one of two conditions: 1) upon the completion of a predefined number, T, of transactions since system initialization or 2) when the observed cache hit rate degrades below a specified performance threshold.

The association rule mining process uses three key metrics: The *support* measures the frequency of an itemset (a collection of items) in the dataset, calculated as the fraction of transactions containing it. A minimum support (β) threshold identifies frequent itemsets in a transaction set. From these, rules of the form $A \implies B$ (milk $\implies$ bread) are generated if they meet a minimum confidence (γ) threshold. The *confidence* represents the conditional probability $P(^{B}/_{A})$. To assess if the co-occurrence is more than mere chance, *lift* (ξ) is calculated as $^{Support(A \cup B)}/_{Support(A).Support(B)}$. A lift value greater than 1 indicates a positive correlation, suggesting the rule is potentially interesting. Selecting an appropriate value is critical: too high and valuable patterns are missed, too low and the process becomes computationally expensive, often yielding numerous insignificant rules. The challenge lies in finding a β range that efficiently produces rules which are not only frequent (support) and reliable (confidence) but also interesting (lift > 1). The following heuristic aims to identify such a β range iteratively:

1. Initial Support Range: Define an initial search range for the β parameter. The lower bound is set to the average support across all individual items in the dataset [74]. The upper bound is set to the maximum support observed for any single item.

2. Discrete Support Range: Select N equidistant candidate β values within the established range $[\beta_{lower}, \beta_{upper}]$.

3. Iterative Rule Generation and Evaluation: Iterate through the N candidate β values, starting from the smallest value and increasing towards the highest. For each candidate β_i

   (a) Generate frequent itemsets $fqItems_i$

   (b) If $|fqItems_i| > 0$, generate association rules $arules_i$ from $fqitems_i$ using a predefined, fixed γ threshold.

   (c) Filter the generated rules, retaining only those with lift, $\xi > 1$, say $arules'_i$

   (d) If $|arules'_i| = 0$, terminate the iterative process, as there are no more significant rules being generated.

   (e) Compute average lift (ξ_{avg_i}) and median lift (ξ_{median_i}) for $arules'_i$, if rules are present.

(f) Compare ξ_{avg_i} and ξ_{median_i} with previous iteration values ($\xi_{avg_{i-1}}$ and $\xi_{median_{i-1}}$, terminate the iterative process if either difference is greater than a threshold. This condition aims to halt the search when the quality (strength of association) of the discovered rules begins to decline notably.

(g) Record the range of β values processed successfully before any termination condition was met. Let the lowest and highest such values encountered be β_{min} and β_{max}.

4. Upon completion of the iteration, define the final candidate range as $[\beta_{min}, \beta_{max}]$. Divide this range into K equidistant points. These K values are proposed as the candidate β thresholds for generating potentially relevant association rules.

Finally, generate the associations rule sets for each of the K minimum support values for a fixed minimum confidence and store them.

For the dynamic deployment of the K generated association rule sets, an adaptive strategy prioritizing precision and stability is adopted. Initially, the Prefetching engine is configured to utilize the rule set derived from the β_{max}. This approach leverages rules based on the most frequent, and presumably most reliable, co-occurrence patterns. It mitigates the risk of cache pollution associated with rules derived from less frequent or potentially spurious patterns. In the event of cache hit degradation, the engine adaptively transitions to the rule set generated with the next highest minimum support threshold. This process repeats iteratively, allowing the system to progressively incorporate rules representing less frequent patterns. Once, we reach β_{min} and the cache degradation still exists, then the Prefetch engine generates new association rulesets based on the last T transactions.

4.4 Evaluation

4.4.1 Experimental Setup

The ASTRA Prefetching framework is implemented in C++ and uses SimCpp20 [75] for discrete event simulation. The association rule generation is implemented in Python using mlxtend library [76]. Our experimental testbed utilizes a 64GB Intel Xeon E5-2620 with 24 cores. Evaluation is performed on four synthetic datasets and a real-world

retail dataset [77].

4.4.2 AR Workload Simulator Design

To simulate multi-user augmented reality interactions in mentioned scenarios (Section 4.2), for which public datasets are unavailable, we constructed a workload simulator. This simulator integrates four key modules to generate realistic user behavior patterns:

- Environment Generation: A 2D map is first created from 3D map data [78]. Within this map, structures like buildings and obstacles are represented as closed polygons, defining the intervening areas as accessible paths for users.
- Transaction Synthesis: The Transaction Generator defines 'N' items within categories (like dairy, vegetables), forms random 'L' sized itemsets and analyzes them to find category relationships for later item placement. It then generates synthetic datasets (DS30-DS75), simulating user lists with varying initial itemset frequencies (e.g., 30% for DS30). To enhance realism, random noise items are added to transactions, which may affect final itemset support.
- Item Placement: The Item Placement module positions items within the 2D environment along predefined polygon boundaries. Items are placed 5-7 units apart, following the category order determined by the Transaction Generator to mimic real-world item access.
- User Path Simulation: This module creates realistic user paths, starting with a Voronoi-based path network derived from item locations and random points in empty spaces. Each simulated user receives a transaction. Their path originates from a common entry/exit point and connects the necessary Voronoi segments to sequentially visit all items in their transaction. To simulate pauses, random 'keyframes' with small associated delays are added along the path.

These generated user paths are used within the main simulation. User arrivals at the environment's entry point are modeled using a Poisson distribution. Although the itemset fraction increases from DS30 to DS75, these synthetic datasets should be treated independently, as the specific combination of itemsets and noise items alters the transaction

structure in each case. In addition to the synthetic data, our experiments incorporate a grocery transaction dataset from [77].

4.4.3 Baselines

We consider two categories of baselines:

1. Caching baselines: We first integrate ASTRA with common caching policies (Least Recently Used(LRU), Least Frequently Used (LFU), First In First Out (FIFO), Popularity(POP)) to understand the impact of our prefetching algorithm. Then we select one of them as the underlying cache mechanism to compare with current prefetching algorithms.

2. Prefetching baselines: There are two main types of prefetching algorithms mostly used in MAR in the literature:

    - Path-based prefetching: It employs a spatial partitioning technique, dividing the environment into uniform cells, and predicts future user locations based on recent movement patterns [67].
    - Intent-based prefetching: It utilizes a basic Markov model to predict the next object to be prefetched. The model has a warm up time before starting to identify relevant prefetches. [72]

4.4.4 Simulation parameters

Interactive AR at 30 FPS requires end-to-end latency below 33.3 ms to prevent lag. Our simulation experiments utilize measurements from Jaguar [1], with parameters specified in Table 4.1. Client-side processing involves downscaling frames to 10 KB before transmission and rendering 3D augmentations upon reception [1]. We model network bandwidths using typical values: 20 Mbps upload / 400 Mbps download for the client (5G-like [79]) and 1 Gbps for the edge-to-cloud link (AWS Direct Connect [80]). Based on analyzing the Objaverse dataset [81], where 70% of 100K objects are under 5MB, we set the object (glb) file size range to 1-5MB for our experiments.

Table 4.1: Latency measures from Jaguar [1]

Task	Execution time (ms)
Pre-processing on client	6.59 ± 1.21
Data transfer to edge	3.9 ± 1.64
Feature extraction, object recognition and matching	14.87 ± 2.04
Post processing on client	2.68 ± 1.32

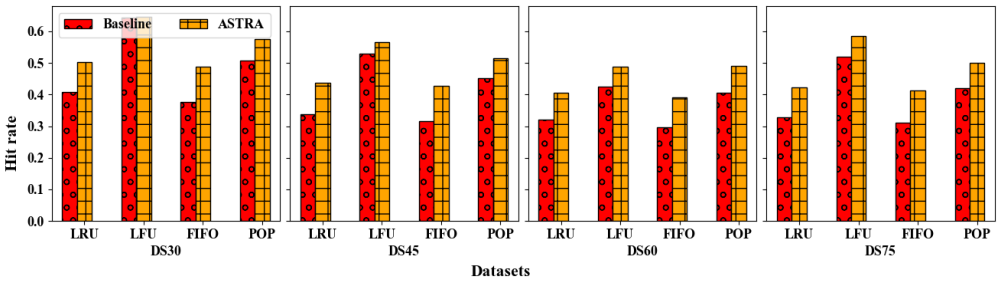


Figure 4.3: Best hit rate achieved by ASTRA in comparison to baseline cache policies

4.4.5 Impact of ASTRA on Caching Policies

This simulation experiment evaluates the ASTRA prefetching strategy integrated with baseline cache algorithms (without minimum support tuning). It uses 1000 users, 100 items, and a cache sized at 10% of total AR asset storage. The key ASTRA parameters (window=25, association threshold=1, proximity threshold=30) were empirically determined and held constant unless stated otherwise. Figure 4.3 shows ASTRA's peak cache hit rates across synthetic datasets with varying minimum support/confidence (30-75%) configurations.

Figure 4.3 and Table 4.2 demonstrate that ASTRA significantly improves hit rates over baseline methods, with gains up to 35%. The improvement is notably lower when combined with the LFU policy. This occurs because LFU prioritizes evicting low frequency items. ASTRA's prefetched items enter with minimal frequency and are thus prone to premature eviction before potential use. Other policies like LRU, FIFO, and POP do not inherently suffer from this issue.

LFU's reliance on historical access counts makes it slow to adapt when item popularity changes. Both LRU (based on access recency) and windowed POP (based on recent frequency) respond better to temporal shifts. Because windowed POP evaluates

Table 4.2: Percentage hit rate improvement of ASTRA over baseline

Dataset \ Cache	LRU	LFU	FIFO	POP
DS30	23.67	0.51	29.35	13.79
DS45	29.19	6.96	34.64	14.15
DS60	26.71	14.92	32.7	21.03
DS75	29.07	12.61	32.89	18.77

recent frequency, it adapts more quickly than LRU to sudden popularity increases. We therefore selected the windowed POP algorithm for subsequent experiments due to its responsiveness to recent access patterns.

4.4.6 Comparison to Current Prefetchers

In the following experiments, we compare POP, Path and Intent based approaches to ASTRA.

Cache hit rate: Figure 4.4 compares the cache hit rates of ASTRA against baseline

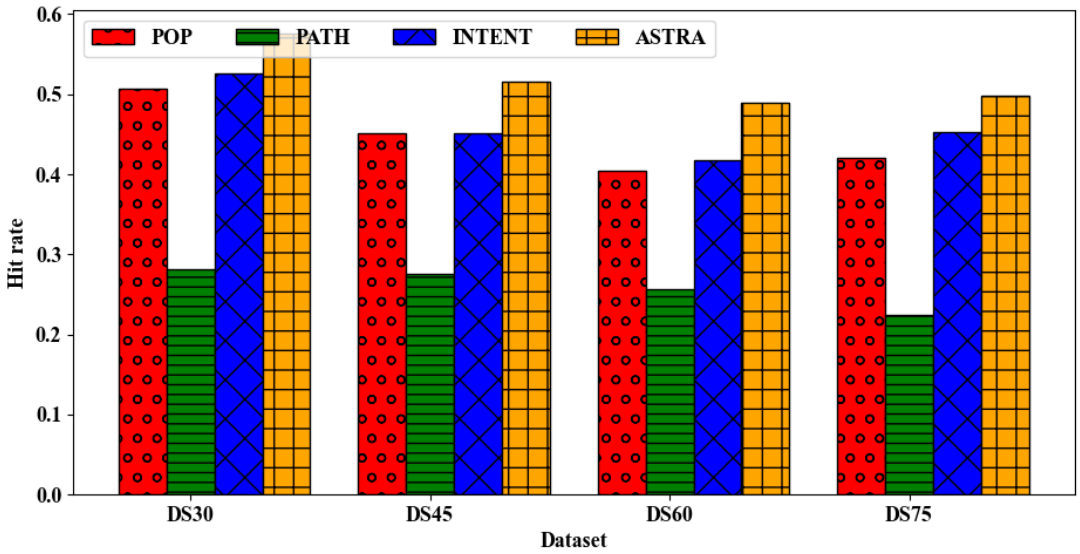


Figure 4.4: Cache hit rate of ASTRA compared to prefetching algorithms. ASTRA improves hit rate by 122%, 17% and 21% compared to PATH, INTENT and POP

methods: Path-based prefetching (PATH), Intent-based prefetching (INTENT), and the POP cache, across the four synthetic datasets. PATH performs adequately only when the spatial distribution of objects is sparse. In denser environments, its trajectory-based approach tends to pollute the cache with objects unlikely to be accessed, leading to hit rates lower than the POP cache baseline. INTENT's predictions rely heavily on the immediately preceding user interactions, often neglecting broader historical context and resulting in suboptimal predictive accuracy. In contrast, ASTRA consistently achieves higher hit rates by integrating contextual user information with a lazy prefetching mechanism, where AR assets are fetched only upon entering user proximity with association relevance. Quantitatively ASTRA improves the average hit rate by 122%, 17%, and 21% relative to PATH, INTENT, and POP, respectively.

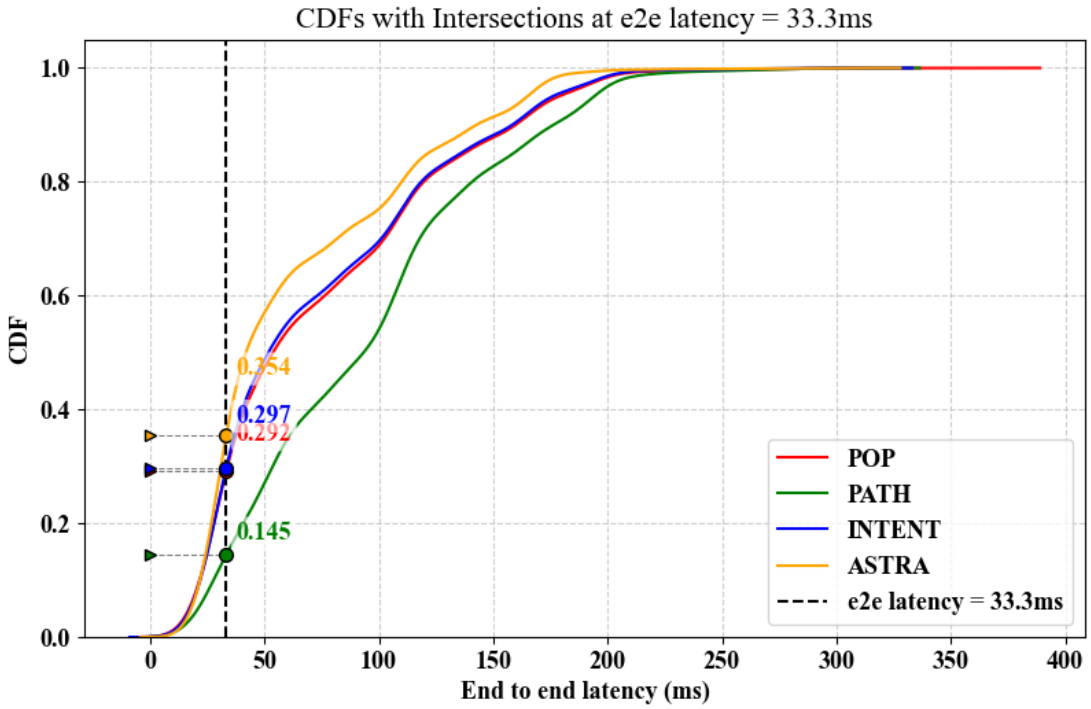


Figure 4.5: End-to-end latency CDF. 35% of the requests are delivered with 33.3ms by ASTRA compared to the 29.7%, 29.2% and 14.5% by INTENT, POP and PATH respectively.

End to end latency and prefetch overhead: Figure 4.5 presents the end-to-end latency CDFs for the prefetching strategies. ASTRA meets the 33.3 ms AR latency requirement

for 35% of requests, surpassing INTENT (29.7%), POP (29.2%), and PATH (14.5%). It also achieves the highest local satisfaction rate (serving 53% of views without cloud interaction), compared to 44% for INTENT, 42% for POP, and 19.5% for PATH. This enhanced local service directly results from ASTRA's superior cache hit rate.

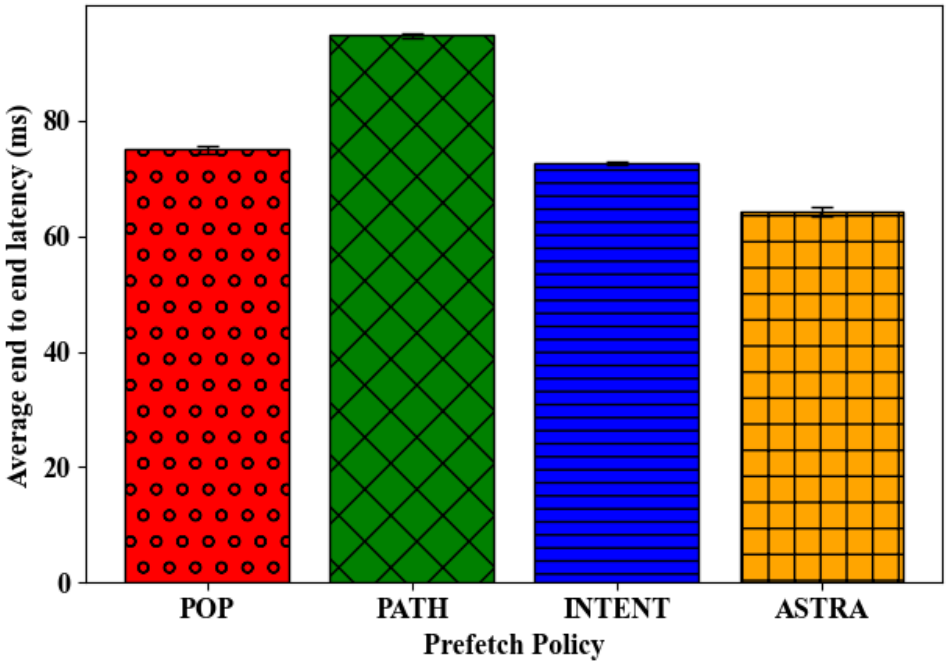


Figure 4.6: Average end-to-end latency. ASTRA achieves 14%, 32% and 11% lower end-to-end latency compared to POP, PATH and INTENT respectively.

Figure 4.6 shows the average end-to-end latency for each prefetching strategy. ASTRA achieves the lowest average latency, directly resulting from its favorable latency distribution, which completes more requests within shorter time bounds (e.g., 60% under 50ms). Specifically, ASTRA reduces average end-to-end latency by 14% compared to POP, 32% compared to PATH, and 11% compared to INTENT.

We calculated the prefetch overhead in MBs across PATH, INTENT and ASTRA for the above experiment. PATH incurred 260 times more data transfer (1190689 MB) than ASTRA (4533 MB) due to aggressively prefetching all assets at predicted locations. As we mentioned earlier, PATH based prefetching is suitable in situation where there

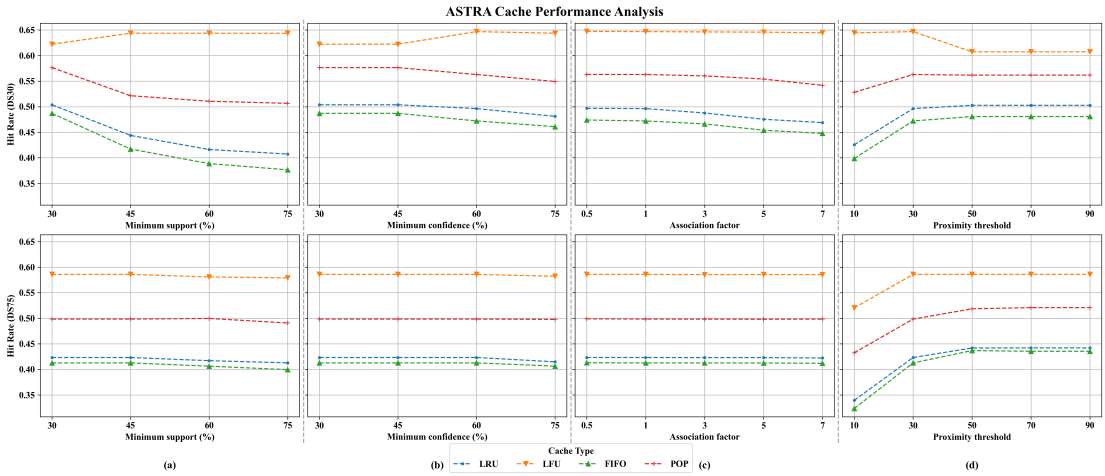


Figure 4.7: ASTRA Parameter analysis: Row 1 and 2 corresponds to DS30 and DS75 datasets respectively. For each parameter shown, it can be seen that for DS30, the change in hit rate is more evident compared to DS75. As the support for the itemsets are high in DS75, the effect of association is diminished. However, proximity helps to reduce the cache pollution with objects currently far away from users.

is object density in a region is low. INTENT had slightly less overhead than ASTRA (3,661 MB, 0.8x) by predicting only the next asset. ASTRA prioritizes fetching AR assets immediately required in the current FoV for rendering from the cloud (on a miss). The associated assets are prefetched in the background to reduce the latency. While INTENT incurs marginally less overhead by prefetching only a single object, ASTRA's slightly higher overhead (prefetching potentially multiple associated items) yields substantially better cache hit rates and lower end-to-end latency, representing a favorable trade-off.

4.4.7 Parameter analysis

Minimum support: This experiment investigates the impact of minimum support on ASTRA's performance across all cache policies. We varied the minimum support threshold from 30% to 75% while keeping other parameters constant (minimum confidence at 30%). Figure 4.7(a) shows the hit rate variation for DS30 and DS75, with similar trends observed for DS45 and DS60. This behavior is consistent across different minimum confidence values. As a critical parameter in association rule mining, higher minimum

support thresholds result in fewer frequent itemsets and consequently, fewer association rules, potentially leading to lower hit rates.

Minimum confidence: This experiment analyzes the influence of minimum confidence on ASTRA's performance. We varied the minimum confidence threshold from 30% to 75% while keeping other parameters constant (minimum support at 30%). Figure 4.7(b) shows this effect on DS30 and DS75, with similar trends observed for DS45 and DS60 across different minimum support values. As a key parameter in association mining, increasing the minimum confidence threshold initially improves or maintains hit rates before causing a decline. This trend reflects the trade-off between rule quality and quantity. Higher thresholds reduce the number of rules, improving quality but potentially excluding relevant associations.

Association factor: This experiment examines the impact of the association factor on ASTRA's cache hit rates. With minimum support and minimum confidence fixed at 30% and 30% respectively, the association factor was varied from 0.5 to 7. Figure 4.7(c) illustrates the results. As dataset support increases (i.e., associated objects are accessed more frequently), the influence of the association factor on identifying relevant objects diminishes. For DS30, the highest hit rate is achieved with an association factor of 1, demonstrating a clear impact. However, for DS75, where frequent itemsets have higher support, the association factor has a less pronounced effect. This suggests that the association factor plays a more critical role in scenarios with lower object access frequencies. Similar observation could be seen in other plots except for proximity threshold.

Proximity threshold: Figure 4.7(d) illustrates the influence of the proximity threshold on cache hit rates for ASTRA-integrated baselines. We varied the proximity threshold from 10 to 90 unit distance while fixing the minimum support at 30%, minimum confidence at 30%, and association factor threshold at 1. The proximity threshold has a significant impact on hit rates. A very low threshold might neglect relevant objects located slightly further away, potentially missing prefetching opportunities. Conversely, an excessively high threshold could lead to prefetching irrelevant objects that are not close enough for immediate user interaction, wasting cache resources. However, it could

be noted that the curves remain constant after certain threshold. This could be attributed to the underlying POP strategy evicting unpopular associated objects. Therefore, selecting an appropriate proximity threshold is crucial for optimizing ASTRA's effectiveness. Similar results are observed for DS45 and DS60.

4.4.8 Minimum support tuning

From Section 4.4.7, it is evident that the parameters associated with the ASTRA prefetching algorithms should be tuned to get the best result. Tuning each parameter to get the optimal result is not a trivial task. So we proposed a heuristic algorithm to tune the minimum support value, which is the entry point to association rule generation, in section 4.3.3. For the following experiments, we set the association factor to 1 and proximity threshold to 30.

Synthetic workload

In this section, we will analyze the cache hit rate performance and end-to-end latency using a mixed synthetic workload and and real world workload.

The mixed synthetic workload is a sequential combination of DS30, DS45, DS60 and DS75 (4000 users). As each dataset has a different itemset that is popular, it is representative of changing workload behavior in the real world.

Cache hit rate: ASTRA without tuning (ASTRA) and ASTRA with tuning (ASTRA-TUNE) use the association rules for DS30 workload initially. The degradation threshold for ASTRA-TUNE is set to 1%. The number of candidate minimum support values for rule generation is 5. It can be seen from Figure 4.8 that ASTRA-Tune detects the change in hit rate and generates new association rules based on 1000 previous transactions or switches to a lower support association rule (red dashed lines). Also, if the association rule is not switched for a period of time (500 transactions), we make a forced select, to see the impact on hit rate. The x-axis represents the FoVs completed across all users. From the figure, it is visible that the ASTRA-Tune is able to improve hit rate most of the time compared to ASTRA. It achieves a maximum gain of 10% over ASTRA and 11% over POP.

End to end latency and overhead: Figure 4.9 shows the CDF for end-to-end latency. ASTRA-TUNE satisfies 33.6% of requests under 33.3ms surpassing the 26.5% and 26%

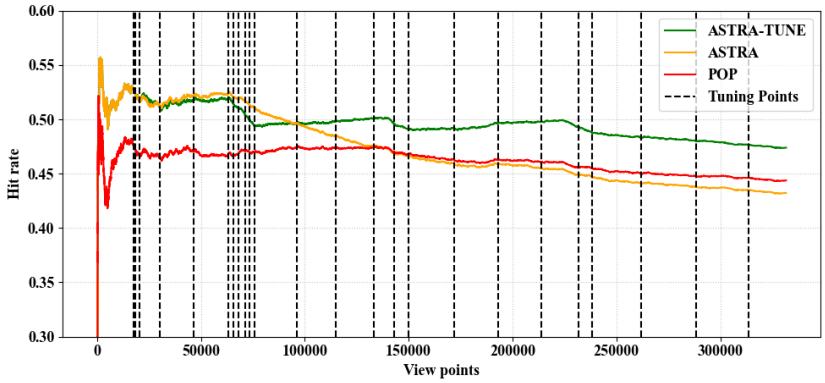


Figure 4.8: Cache hit rate for synthetic workload. ASTRA with tuning achieves higher hit rate amidst change in workload.

of ASTRA and POP respectively. Also, 49% of the requests are satisfied by ASTRA-TUNE under 50ms. The ability of ASTRA-TUNE to adapt to workload changes allows it to improve the end-to-end latency. As for the prefetch overhead, ASTRA-TUNE, has 3.5X (51905 MB) higher data fetching from cloud compared to ASTRA (13870 MB). ASTRA either ends up defaulting back to POP cache policy or fetches assets only when association based on old rules are found, resulting in the lower prefetch overhead.

Real workload

In this section, we will use a real world grocery trace consisting of 9835 transactions. There are 169 unique items in the dataset. The

Cache hit rate: ASTRA and ASTRA-TUNE use the association rules generated with 10% minimum support and 5% minimum confidence for the first 1000 transactions initially. The rest of the transaction is used for the experiment. It can be seen that ASTRA-TUNE achieves a maximum hit rate improvement of 10% and 17% when compared to ASTRA and POP respectively. Initially, because of tuning, ASTRA's hit rate improves (first black dashed line) and then it undergoes rule selection and rule generation when there is a steep dip in the hit rate. Afterwards, the rule selection occurs periodically as explained earlier without a degradation to hit rate.

End-to-end latency and overhead: ASTRA-TUNE deliver 22.4% of the requests under 33.3ms compared to 17.6% and 17.5% of ASTRAand POP respectively. Also,

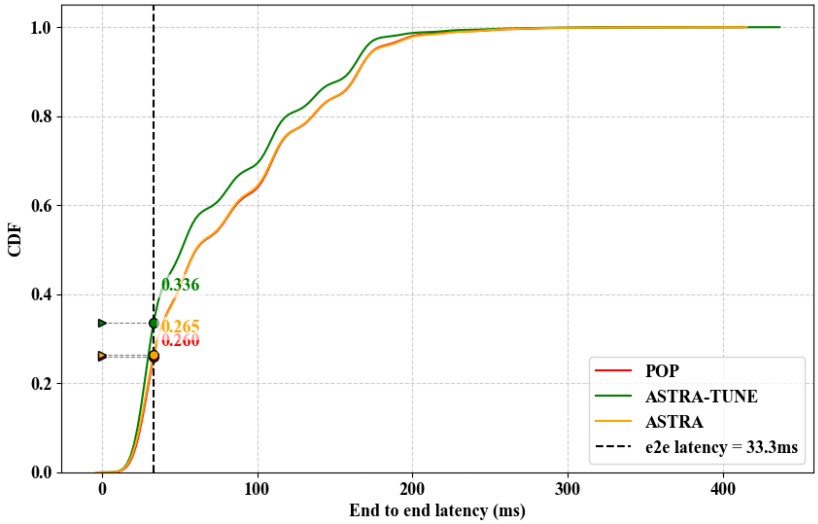


Figure 4.9: End-to-end latency CDF for synthetic workload.

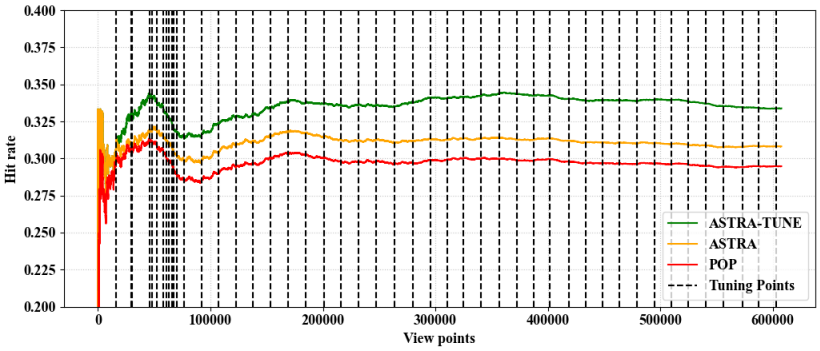


Figure 4.10: Cache hit rate for real workload

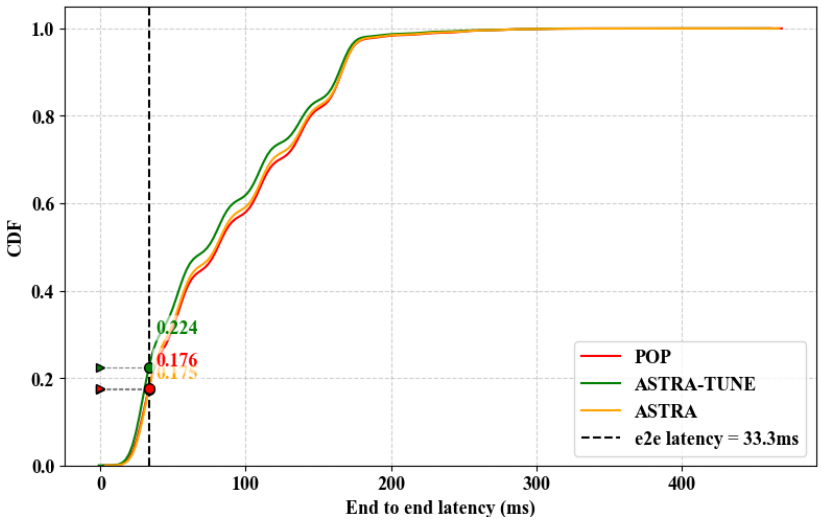


Figure 4.11: End-to-end latency CDF for real workload

ASTRA-TUNE completes 36% of the request under 50ms. As for the prefetch overhead, ASTRA-TUNE has 7.8X (39791 MB) higher data fetching compared to ASTRA (5092 MB).

4.5 Related Work

Mobile augmented reality (MAR) applications [82] enhance user perception by integrating virtual objects into their view. These applications typically employ a distributed architecture across mobile devices, edge servers, and the cloud [83–85]. Efficiently partitioning compute- and data-intensive operations across these tiers is crucial. While MAR applications initially relied on local storage on mobile devices for rapid retrieval and rendering of virtual objects [86], on-demand fetching from the cloud has become prevalent with increasing application complexity [87–90]. However, this approach faces challenges in delivering immersive experiences due to increasing object sizes and high cloud latency.

Edge caching has emerged as a promising solution for AR applications [91, 92], reducing latency and offering greater storage capacity compared to mobile devices. Existing research explores various edge caching strategies. Cachier [93] employs a latency-minimizing model that balances load distribution between cloud and edge, considering

network conditions and request locality. Agar [94] uses dynamic programming to identify popular data chunks for caching. CEDC-O [95] formulates edge data caching as an optimization problem considering caching cost, migration cost, and quality-of-service penalties. While CARS [96] and SEAR [97] utilize Device-to-Device (D2D) communication for sharing cached virtual objects, they lack prefetching mechanisms.

Several recent research efforts focus on leveraging the predictability of user movement within both virtual and physical environments to enhance AR content delivery. By anticipating where a user is likely to look or move next, prefetching strategies [66–68] proactively retrieve relevant AR artifacts onto edge servers, thereby reducing interaction latency. Methods utilizing predicted changes in the user's viewport [69] or Field of View (FoV) [70] or gaze [71] are approaches for guiding this prefetching process.

Other research efforts focus on caching content for different media types at the edge. Works like [98–100] explore caching tiles of 360° videos to enable processing reuse. Space [101] and Leap [102] investigate prefetching video segments for users at the edge. EdgeBuffer [103] leverages user mobility patterns across access points to prefetch data to anticipated locations.

Association rule mining (ARM) [104, 105] has established itself as a valuable technique for prefetching data in various domains, including e-commerce recommendation systems, fraud detection, and social network analysis. For instance, Mithril [106] leverages historical patterns of cache requests within cloud applications to derive item association rules using a variant of ARM called sporadic-ARM. Similarly, web prefetching, which involves caching web objects in anticipation of user requests, is a well-researched area [107]. However, the potential of ARM for prefetching in augmented reality (AR) applications remains largely unexplored.

4.6 Conclusion

This chapter addressed the challenge of reducing latency in compute- and data-intensive Mobile Augmented Reality (MAR) applications by optimizing edge caching. We proposed ASTRA, a novel prefetching policy for edge AR caches that leverages object associations (mined from user interactions), temporal relevance (via an association factor), and spatial proximity (user location relative to objects) to proactively fetch virtual

objects from the cloud to the edge. Through extensive evaluation using both synthetic and real-world workloads, we demonstrate that ASTRA significantly improves cache hit rates compared to current prefetching algorithms, achieving cache hit rate gains of up to 35% and reducing average end-to-end latency by up to 14% compared to baseline prefetching algorithms. Furthermore, the proposed adaptive tuning algorithm for association rule minimum support demonstrated its ability to handle dynamic workloads, further improving ASTRA's cache hit rate by an additional 10%. These findings highlight the potential of context-aware, application-specific prefetching to substantially enhance user experience in demanding edge applications like MAR.

4.6.1 Chapter Takeawys

⇒ *Context is Key for AR Prefetching*: Generic or purely movement-based prefetching is insufficient for MAR. Leveraging application-specific context, such as object associations derived from user interactions and spatial proximity, significantly improves prediction accuracy and cache performance.

⇒ *Balancing Recency and Frequency Matters*: Incorporating temporal relevance (recency) alongside association frequency (support/confidence) allows the prefetcher to adapt more quickly to short-term trends in user behavior.

⇒ *Proximity Enables Resource Efficiency*: A lazy fetching strategy triggered by user proximity prevents unnecessary cache pollution and bandwidth consumption by only fetching associated objects when they are likely to enter the user's immediate field of view.

⇒ *Adaptivity Handles Dynamic Workloads*: Pre-computed or periodically updated association rules can become stale. An adaptive tuning mechanism that adjusts rule generation parameters (like minimum support) based on observed performance is crucial for maintaining high hit rates under changing user behavior patterns.

Chapter 5

Viveka: Adaptive sampling and selection of Sensors in Smartwatches for Human Activity Recognition

5.1 Introduction

The proliferation of Internet of Things (IoT) from ubiquitous smart cities to precision agriculture demonstrates how these multi-sensor systems are deeply embedded in our daily life. They drive applications ranging from daily fitness tracking [108] and animal migration monitoring [109] to high stakes scenarios such as cargo crew wellness assessment [110], military personnel health monitoring during active missions [111] and predictive maintenance in industrial manufacturing facilities [112]. In all these domains, the trend is towards increasing number of sensors, for instance in Body Sensor Networks (BSNs) or Industrial IoT (IIoT) clusters, to capture high fidelity data.

The potential of these multi-sensor systems are impeded by the limited battery/ strict energy budgets, which has not kept pace with the growth of sensor capabilities and processing demands. Figure 5.1 shows the battery life in days (1-14) for known commercial smart wearables. Continuous high fidelity sensing is the worst case scenario

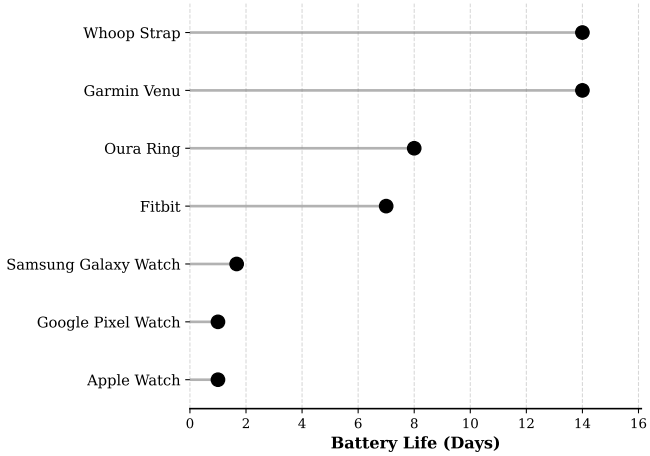


Figure 5.1: Battery of commercial smart wearables.

for power consumption, leading to rapid battery depletion and frequent data gaps during recharge cycles. These gaps could result in critical event misses relevant to the application at hand. Furthermore, the deluge of data from always-on sensors overwhelms an already limited on-device storage on the sensors systems.

To mitigate these constraints, existing research on energy optimization focus on two isolated strategies: Sensor selection and Adaptive sampling. In Sensor selection, a subset of sensors from a global set are identified to reduce active hardware components [113–118]. As for Adaptive Sampling, the frequency of data collection per sensor is varied to lower the computation and sensing load [119–123]. While both these approaches are effective, combining them could reap the compounding benefits. Existing joint optimization strategies typically rely on instance-wise reinforcement learning agents [124], early-exit neural architectures [120] or static model pruning [125]. However, these approaches often introduce significant operational penalties, including energy-inefficient sensor thrashing (sensors toggling ON/OFF), unpredictable computational overhead or an inability to adapt to instantaneous context changes.

In this paper, we propose Viveka, a context-aware framework that achieves high energy savings by aggressively pruning data generation at the source. Operating on the insight that the sensing requiements are dictated by the user's context, Viveka uses signal processing techniques to create a dynamic sensing signature for every unique context. First, Viveka employs Permutation-based Feature Importance to identify the

minimal sensor set required for each context. Second, it utilizes spectral energy analysis to tailor the sampling fidelity to the biomechanical rigor of the movement, ensuring high rates are only used when the context demands it. This results in a granular lookup table that allows the device to dynamically adapt its hardware configuration to the user's context.

We validate Viveka using Human Activity Recognition (HAR) [126] on smart wearables as a primary tested. HAR presents a challenging usecase due to the stochastic nature of human motion and the size/power constraints of wearable hardware. However, the principles of Viveka is generalizable to any energy constrained, multi-sensor IoT environment.

The main contributions of this paper are as follows:

- We formulate the joint optimization of sensor selection and sampling rates as an NP-Hard energy minimization problem constrained by classification error.
- We propose Viveka, a framework that solves this by dynamically mapping hardware configurations to the user's semantic context, shifting from a one-size-fits-all sensing policy to an adaptive regime.
- We introduce a two-stage heuristic that uses Permutation-based Feature Importance (PFI) to identify context-specific sensor sets and spectral energy analysis to determine the lowest viable sampling rate per activity.
- We perform an extensive evaluation on two real-world datasets, demonstrating that Viveka achieves 63-70% energy savings and 75-78% data reduction compared to standard baselines, while maintaining classification accuracy within 3-5% of the theoretical upper bound.
- We acknowledge that Viveka's reliance on offline analysis assumes relatively stable contexts, which may required periodic tuning in highly dynamic environments. Currently we use a fallback mechanism to ensure robustness. To support further exploration, we will release the artifacts upon acceptance.

5.2 Motivation

In this section, we explore various wearable applications that benefit from extended battery life and the application-specific insights that can be leveraged to achieve it.

5.2.1 Motivating Applications

The need for extended battery life is essential to realize the full potential of wearable technology. Consider the following applications where extended battery life is critical:

- **Daily Fitness and Wellness Tracking**: The consumer wellness market represents the most widespread use of wearable technology. Devices that track daily fitness, sleep quality and stress levels rely on consistent, long-term use to provide meaningful insights. The user experience is directly tied to the device's convenience and a primary point of friction is the need for frequent charging. A user who must charge their watch every day may forget to put it back on, resulting in incomplete data for activity goals and a total loss of sleep tracking for that night. To become a seamless part of a user's life, these devices must demand minimal attention. Extended battery life is therefore a key feature that directly impacts user retention and the overall effectiveness of the wellness platform [127].

- **Continuous Chronic Disease Monitoring**: In healthcare, wearables are transforming the management of chronic conditions such as cardiovascular disease, diabetes and epilepsy. These devices enable continuous, real-time monitoring of vital signs like heart rate, blood glucose levels and motion data, allowing for early detection of adverse events and providing a rich dataset for personalized treatment plans. For these applications to be effective, patient adherence is critical and data continuity from the wearable is paramount. Frequent charging requirements create a significant burden, especially for elderly patients and lead to gaps in data where a critical event, like an arrhythmia or a seizure, could be missed. An extended battery life that allows a device to operate for weeks or months without intervention makes the system more practical, less obtrusive, and more reliable for safeguarding patient health [128].

- **Crew Readiness Monitoring in Critical Missions**: Monitoring the operational readiness of personnel in austere environments, such as on oil rigs or during military missions, is critical for safety and mission success. In these remote and demanding settings, frequent charging is not just inconvenient but also introduces a significant point of failure. A device with a depleted battery cannot provide vital alerts on fatigue or stress, directly compromising crew safety. To be effective, these wearables must operate for extended periods with minimal user intervention. This necessitates advanced, on-board energy management systems that can balance data collection with power preservation to ensure reliability [110].

Across these applications, activity recognition is a common thread. In all three cases, the primary goal is to capture and analyze activity patterns to extract meaningful insights. However, the energy constraints of wearable devices make this goal challenging to achieve. To address this challenge, researchers have proposed various techniques to optimize energy consumption while maintaining the accuracy of the activity recognition system, including the sensor selection and adaptive sampling rate methods discussed previously.

5.2.2 Sensor energy-accuracy trade offs

The effectiveness of sensor selection arises from the significant variance in power consumption across different sensors. Consider a typical set of commercial sensors used in a smartwatch [129], with specifications taken from their datasheets: the BMI270 (accelerometer and gyroscope), BMM350 (magnetometer) and BMP858 (pressure sensor). As shown in Figure 5.2, the current consumption for these sensors operating at a comparable low-power sampling rate (near 50 Hz) varies dramatically. The gyroscope is the most power-hungry, followed by the magnetometer, pressure sensor and accelerometer. In this scenario, if the gyroscope's data could be omitted for certain activities without a significant loss in HAR accuracy, the current consumption for sensing could be reduced by approximately 70%. This highlights the need to identify the optimal set of sensors required for each activity to maximize energy savings.

To gain better insight into the relationship between activities and sensors, we conducted an experiment where a Long Short-Term Memory (LSTM) model was trained on

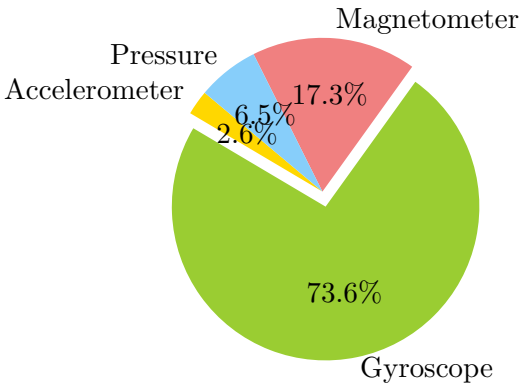


Figure 5.2: Current consumption of sensors at 50Hz

the WISDM [130] dataset. The WISDM dataset consists of data from smartphones and smartwatches across multiple users for 18 activities. Since our focus is on wearables, only the smartwatch data was used for training. The data consists of accelerometer and gyroscope data sampled at 20 Hz. We then applied Permutation Feature Importance (PFI) [131] to analyze the contribution of each sensor feature to each activity classification. Figure 5.3 shows the accuracy drop when a feature is permuted, indicating the relative importance of each sensor for each activity. The x-axis represents the sensor features (accelerometer: acc_x, acc_y, acc_z; gyroscope: gyro_x, gyro_y, gyro_z) and the y-axis represents the activities (A to S). The results show that the importance of each sensor varies by activity. For example, activity A is dependent on both accelerometer and gyroscope data, while activity D is mostly dependent on accelerometer data. This insight can be utilized at runtime to decide which sensors to activate for a given activity, thereby reducing energy consumption.

The sensor sampling rate also provides an opportunity for energy savings, as it can often be reduced without a major impact on performance. To demonstrate this, we extended our previous experiment using the same LSTM model trained on the 20 Hz WISDM dataset. During testing, we downsampled the data to mimic 10 Hz sensing by selecting every other data point. We then interpolated this data back to 20 Hz before feeding it to the model. This resulted in the overall test accuracy dropping by only 3.66%, from 84.11% to 80.44%. Figure 5.4 shows the permutation-based accuracy drop for the 10 Hz data. When compared to the original 20 Hz results, there is only a slight

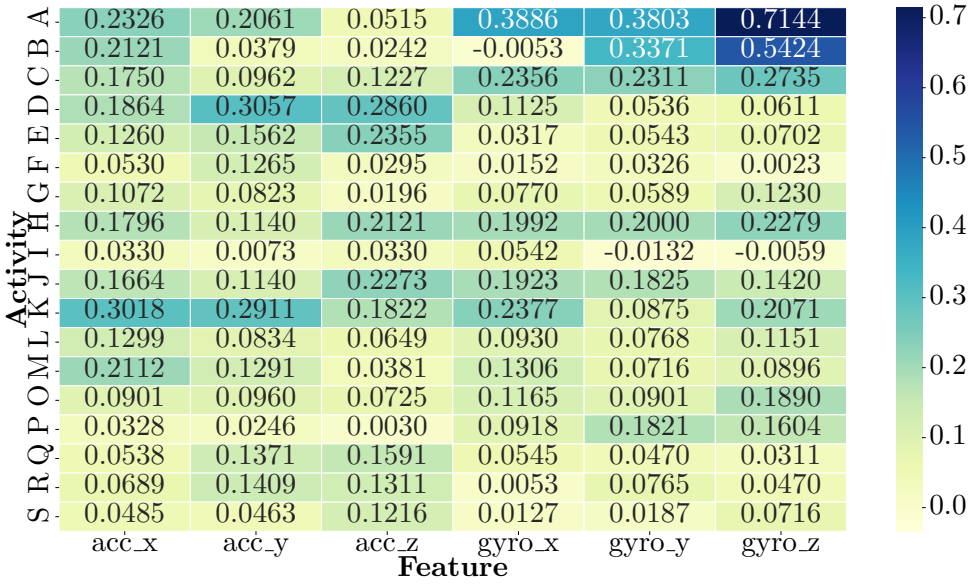


Figure 5.3: PFI for the WISDM dataset at 20Hz

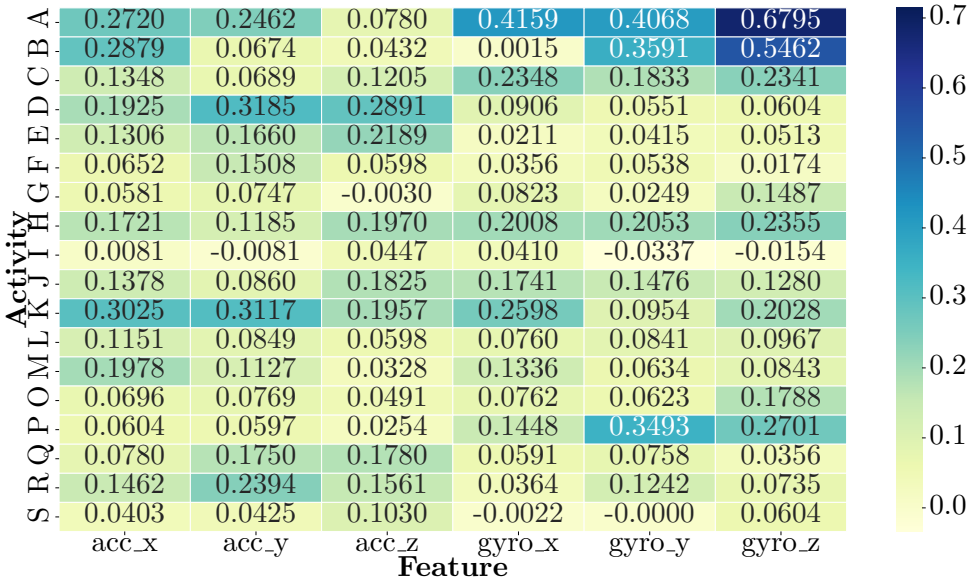


Figure 5.4: PFI for the WISDM dataset at 10Hz

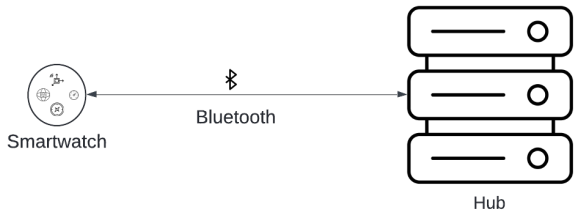


Figure 5.5: System architecture

change in the accuracy drop for most activities, with the notable exception of gyro_z for activity A. In addition, it could be also see that the drop in accuracy slightly increases in 10 Hz. This is due to the interpolation process resulting in smoother data. This shows that even at a lower sampling rate, the model can still achieve high accuracy, highlighting the potential of adaptive sampling to reduce energy consumption while maintaining classification performance.

5.3 Preliminary Investigation: Validating Activity-Aware Adaptation

Before developing the full Viveka framework, a preliminary investigation was conducted to validate the core concepts of activity-aware adaptive sampling and sensor selection using simpler heuristics. This initial study served as a crucial proof-of-concept, demonstrating the potential for significant energy savings in HAR tasks on wearables.

5.3.1 Methodology

This preliminary system employed a wearable-hub architecture (Figure 5.5, Figure 5.6). Key aspects included:

1. Activity-Triggered Adaptive Sampling: The wearable adjusted its sampling rate from a predefined discrete set ([1, 2, 5, 10, 25, 50] Hz). The rate increased upon detection of an activity change (inferred by an LSTM model on the hub) and decreased during periods of stable activity, aiming to capture transitions accurately while saving power during steady states.

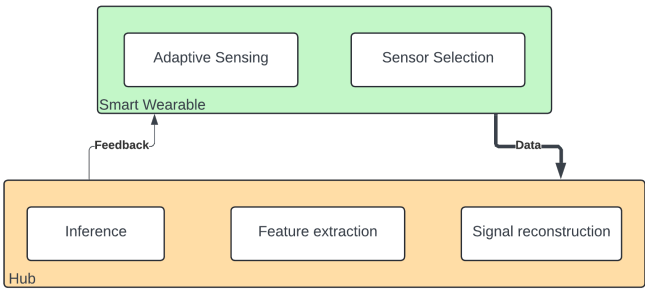


Figure 5.6: Modules within smartwatch and hub

2. Offline Power-Based Sensor Selection: An offline analysis was performed using historical data. Sensors were ranked by power consumption (Figure 5.7). A zeroing-out method was used to iteratively remove the highest-power sensors, retraining the HAR model and evaluating the impact on accuracy. Sensors whose removal caused minimal accuracy degradation were identified for potential deactivation during runtime.

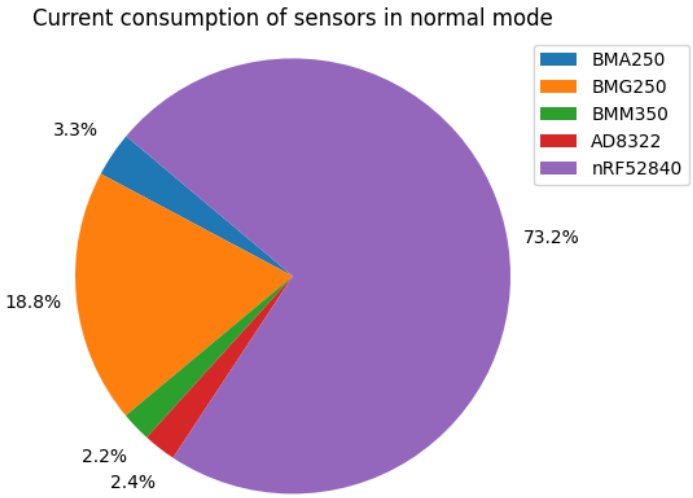
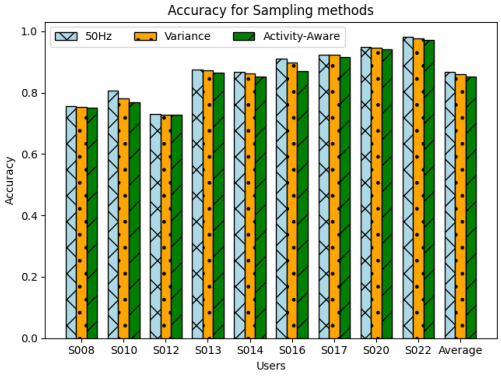
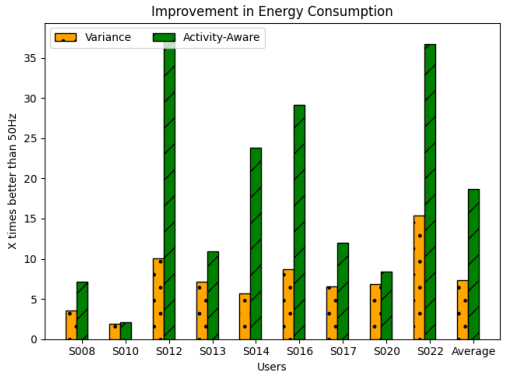


Figure 5.7: Current consumption of sensors for a single user of MHEALTH dataset. BLE followed by gyroscope consumes majority of the current during continuous sensing.



(a) Inference accuracy for HARTH. Variance and Activity aware sampling achieve similar accuracy as baseline frequency for most users.



(b) Sensing energy consumption improvement for HARTH. The number of samples in Activity-Aware is much less than Variance, resulting in much low sensing energy consumption.

Figure 5.8: HARTH results

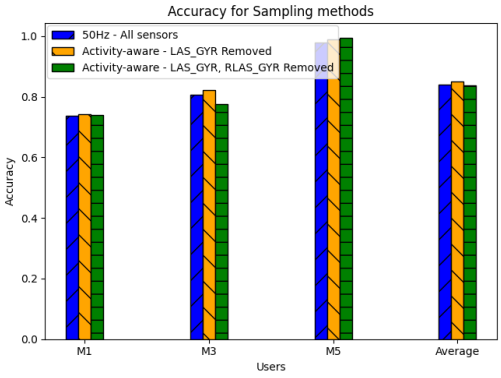
3. Hub-Based Processing: Sensor data reconstruction (linear interpolation), feature extraction, and HAR inference (LSTM) were performed on a nearby hub, with feedback sent to the wearable via BLE. BLE was duty-cycled to conserve transmission energy.

5.3.2 Key Findings

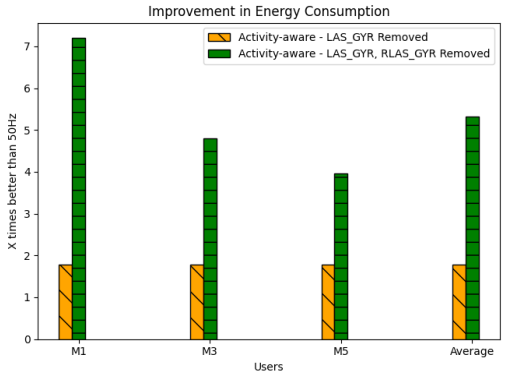
Evaluation using simulations on the HARTH and MHEALTH datasets yielded promising results:

1. Adaptive Sampling Efficacy (HARTH): The activity-aware adaptive sampling strategy achieved an average 18.6X reduction in sensing energy compared to continuous 50Hz sampling, significantly outperforming a variance-based adaptive method (7.3X). This substantial saving was achieved with minimal impact on HAR accuracy (Figure 5.8a, Figure 5.8b).

2. Sensor Selection Viability (MHEALTH): Removing high-power sensors (gyroscopes) identified through the offline analysis resulted in an average 5.1X reduction in sensing energy on the MHEALTH dataset, again with negligible loss in HAR accuracy (Figure 5.9a, Figure 5.9b).



(a) Inference accuracy for MHEALTH. Removing the gyroscopes has little impact for the inference result of MHEALTH



(b) Sensing energy consumption improvement for MHEALTH. With the removal of gyroscopes, the energy consumption improved in the range 3.9-7.2X compared to baseline frequency.

Figure 5.9: HARTH results

3. Combined Energy Savings: When combining adaptive sampling, sensor selection (gyroscope removal), and BLE duty-cycling, the system demonstrated an overall average energy saving of 4.1X compared to a baseline of continuous 50Hz sampling with always-on BLE (Figure 5.10).

5.3.3 Implications

This preliminary study successfully demonstrated that leveraging activity context for both sampling rate adaptation and sensor selection offers significant potential for extending wearable battery life. The 4X overall energy saving, achieved without substantial accuracy loss, validated the core hypothesis and motivated further research into more sophisticated, dynamic, and integrated co-optimization strategies. However, this initial approach also highlighted key limitations, namely:

1. Hub Dependency and Communication Overhead: The adaptive sampling mechanism relies entirely on activity inference performed on an external hub. This necessitates continuous BLE data transmission and feedback, consuming significant energy and proving unreliable in scenarios with intermittent connectivity. A truly autonomous and robust wearable system requires more on-device intelligence.

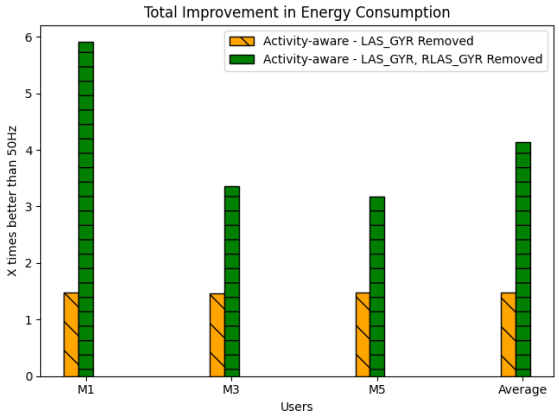


Figure 5.10: Total improvement in energy consumption. The total enery improvement when sensing and BLE is combined. We are able to achieve 3.1-6X improvement compared to baseline frequency.

2. Static, Offline Sensor Selection: The sensor selection process is performed offline based on historical data and power profiling. This results in a static configuration that cannot adapt to dynamic changes in context or sensor relevance during runtime. The optimal sensor subset for an activity might change based on environmental factors or even the current sampling rate, which this offline approach cannot capture.

3. Lack of Integrated Co-Optimization: The preliminary work addressed adaptive sampling and sensor selection largely as separate problems. However, these two aspects are deeply interdependent; the optimal sampling rate might change depending on which sensors are active, and the most informative sensors might differ at various sampling rates. Treating them independently leads to suboptimal energy-accuracy trade-offs.

4. Reliance on Simple Heuristics: The adaptive sampling trigger (activity change detection) and sensor removal method (power-based zeroing-out) are relatively simple heuristics. While effective as a proof-of-concept, they may not fully capture the complex relationship between sensor data, sampling rates, specific activities, and the resulting inference accuracy.

These limitations highlight the need for a more sophisticated, integrated, and on-device strategy. To achieve truly optimal energy efficiency while maintaining high accuracy, the system must dynamically co-optimize both the active sensor set and their individual sampling rates in real-time, based on a deeper understanding of the specific informational requirements of the inferred user activity. This motivates the development of the Viveka framework, which employs advanced techniques for context-aware co-optimization directly on the wearable device.

5.4 System Model

In this section, we describe the system model for our work. Based on this model, we formulate the sensor and sampling rate selection as an energy minimization problem.

5.4.1 System Components

The system we consider consists of multiple sensors that reside on a single edge device, such as a smartwatch, ring, or band. Alternatively, the system could be a body sensor network (BSN) where sensors connect to a central gateway device. The edge device is constrained by limited battery capacity, processing power (CPU/GPU), and storage. It is powered by a battery of limited capacity, necessitating periodic recharging. Data collected from the sensors is processed and stored on the device for a period before being transmitted to a nearby edge server (e.g., a smartphone, tablet, or hub) via low-energy communication protocols such as Bluetooth Low Energy (BLE). Upon receiving the data, the edge server reconstructs the original signal using techniques like linear interpolation. This reconstructed data is then fed to a model for activity recognition and the results are utilized by upstream applications. Figure 5.11 shows a high-level overview of the system.

5.4.2 Problem Formulation

We will now formulate the sensor selection and adaptive sampling as an optimization problem based on the system model.

Consider the following parameters for the formulation:

- $\mathcal{S}$: The set of available sensors.

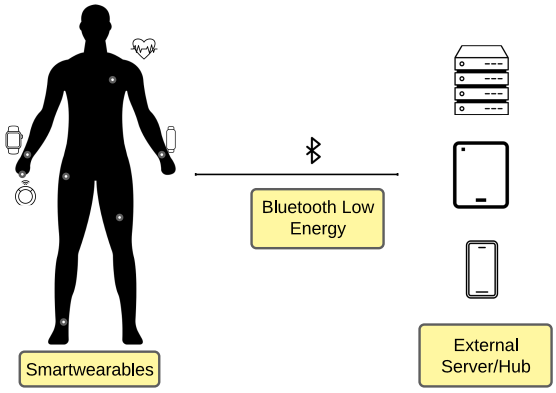


Figure 5.11: System Model

- R_s: The discrete set of sampling rates for sensor $s \in \mathcal{S}$, indexed by j. To account for sensor selection, we explicitly include the zero rate (sensor OFF). Thus, $R_s = \{r_{s,0}, r_{s,1}, \ldots, r_{s,|R_s|-1}\}$, where $r_{s,0} = 0$ represents the inactive state and $r_{s,j} > 0$ for $j > 0$ represents active sampling rates.

- $E(s, r)$: The power consumption of sensor s operating at sampling rate r. We assume $E(s, 0) = 0$.

- $\mathcal{X}$: The set of all feasible configurations. A configuration vector $x \in \mathcal{X}$ represents the sampling rate selection for all sensors, denoted as $x = [x_1, x_2, \cdots, x_{|\mathcal{S}|}]$, where $x_s \in R_s$ is the rate chosen for sensor s. The size of this search space is given by $|\mathcal{X}| = \prod_{s \in \mathcal{S}} |R_s|$. If all sensors have N rates, this simplifies to $N^{|\mathcal{S}|}$, indicating exponential growth with respect to the number of sensors.

- $C(x)$: The expected classification error of the inference model when data is collected using configuration x.

- $C_{threshold}$: The maximum tolerable classification error for the application to remain viable.

Energy and Decision Variables

The total energy consumption for a configuration x is the sum of the energy costs of the selected rates for each sensor:

$$E_{total}(x) = \sum_{s \in \mathcal{S}} E(s, x_s) \tag{5.1}$$

To formulate the optimization problem as an Integer Linear Program (ILP), we introduce binary decision variables $\delta_{s,j}$ to represent the selection of specific rates:

$$\delta_{s,j} = \begin{cases} 1 & \text{if sensor } s \text{ is assigned sampling rate } r_{s,j} \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

These binary variables map to the configuration x such that $x_s = \sum_j r_{s,j} \cdot \delta_{s,j}$. Consequently, the energy function can be rewritten in terms of δ:

$$E_{total}(x) = \sum_{s \in \mathcal{S}} \sum_{j} E(s, r_{s,j}) \cdot \delta_{s,j} \tag{5.3}$$

Optimization Problem (APSSE)

We will call the optimization as *Accuracy Preserving Selection and Sampling for Energy efficiency* (APSSE) problem.

$$\textbf{Minimize:} \quad \sum_{s \in S} \sum_{j} E(s, r_{s,j}) \cdot \delta_{s,j} \tag{5.4}$$

$$\textbf{Subject to:} \quad C(x) \leq C_{threshold} \tag{5.5}$$

$$\sum_{j} \delta_{s,j} = 1, \quad \forall s \in S \tag{5.6}$$

$$\delta_{s,j} \in \{0, 1\}, \quad \forall s \in S, \forall j \tag{5.7}$$

The summations over j iterate through all valid rate indices $\{0, \ldots, |R_s| - 1\}$ for sensor s. The constraint in (5.5) depends on the configuration x, which is determined by the decision variable δ as described above. Constraint (5.5) ensures the solution meets the

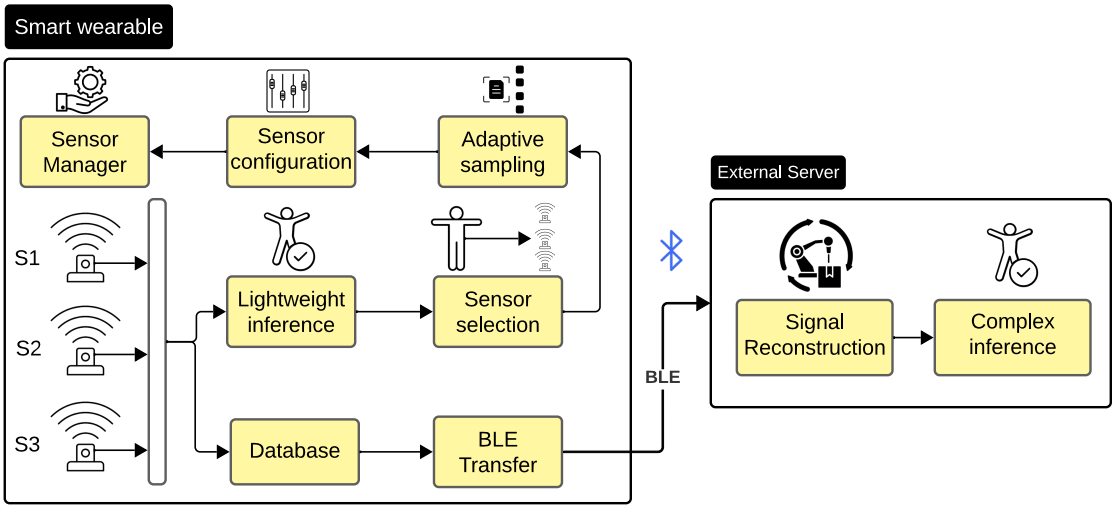


Figure 5.12: Viveka framework

accuracy requirement. Constraint (5.6) enforces that every sensor must be in exactly one state (which includes the state of being OFF/0Hz).

In Appendix A, we prove APSSE is NP-Hard. The computational complexity of the APSSE is exponential, $O(N^{|\mathcal{S}|})$, where N is the number of discrete sampling rates per sensor. For a system with large number of sensors, an exact solver will require a long time to converge, rendering it completely unsuitable for online, per-activity adaptation, thereby necessitating the heuristic approach proposed in the next section.

5.5 Viveka

In this section, we propose Viveka, a novel, context-aware framework that selects sensors and corresponding sampling rates based on the context to reduce the energy consumption associated with sensing. Since the APSSE joint optimization is NP-Hard, Viveka employs a two stage heuristic: first, it determines the minimal set of sensors required per activity. Second, it identifies the sample rate required per sensor in the minimal set while ensuring minimal impact on overall task accuracy.

Figure 5.12 showcases the Viveka framework and its main modules. Our primary contributions lie within the Sensor Selection and Adaptive Sampling modules.

5.5.1 Sensor Selection

The strategy (Algorithm 5) employs an offline analysis to determine the most important sensor set for each distinct activity using Permutation-based Feature Importance (PFI). In our system, we have two pre-trained models: A light-weight model is used for real-time activity inference on the wearable device, while the model on edge server is used for more complex processing. The goal of the light-weight model is to provide an understanding of the current context and we do not expect perfect accuracy. For example, it may report someone is doing an activity similar to running or jogging. To ensure a comprehensive selection, PFI is calculated independently for the two models.

Algorithm 5 Offline Generation of the Sensor Selection Map ($\Pi_{\mathcal{S}}$)

Input: Training data $\mathcal{D}$, models $M_{wearable}$ & M_{edge}, activities $\mathcal{A}$, sensors $\mathcal{S}$, PFI threshold θ_{PFI}
Output: Policy map $\Pi_{\mathcal{S}}$ mapping each activity to its optimal sensor set.
▷ Initialize the policy map
$\Pi_{\mathcal{S}} \leftarrow$ new empty Map()
for each activity $A \in \mathcal{A}$ **do**
    ▷ Filter data for the current activity
    $D_A \leftarrow \{d \in \mathcal{D} \mid \text{label}(d) = A\}$
    ▷ Identify important sensors from both models using PFI
    $S_w \leftarrow \{s \in \mathcal{S} \mid \text{PFI}(M_{wearable}, D_A, s) \geq \theta_{PFI}\}$
    $S_e \leftarrow \{s \in \mathcal{S} \mid \text{PFI}(M_{edge}, D_A, s) \geq \theta_{PFI}\}$
    ▷ Store the union of important sensors
    $\Pi_{\mathcal{S}}[A] \leftarrow S_w \cup S_e$
return $\Pi_{\mathcal{S}}$

In PFI, the features corresponding to a sensor are permuted and fed to the model to measure the impact on accuracy. A significant drop in model accuracy indicates that the sensor's features are highly important. For each activity, we identify the sensors that are important for both models to ensure that no critical information is lost when the data is processed on the edge server. The selection of these sensors is based on a

PFI threshold (θ_{PFI}). The final set of essential sensors for a given activity is the union of the important sensors identified from both models, creating a policy ($\Pi_{\mathcal{S}}$) that is effective across the entire system.

5.5.2 Per sensor Sampling rate Selection

Following the identification of essential sensors ($\Pi_{\mathcal{S}}$) for each activity, the strategy determines the optimal sampling rate for each of these sensors individually (Algorithm 6). This is accomplished by applying the Nyquist-Shannon sampling theorem in an energy-aware context.

Algorithm 6 Offline Generation of the Optimal Sampling Rate Map ($\Pi_{\mathcal{R}}$)

Input: Training data $\mathcal{D}$, Sensor map $\Pi_{\mathcal{S}}$, Activities $\mathcal{A}$, FFT energy threshold θ_{FFT}
Output: Optimal sampling rate map $\Pi_{\mathcal{R}}$.
▷ Initialize the rate policy map
$\Pi_{\mathcal{R}} \leftarrow$ new empty Map()
for each activity $A \in \mathcal{A}$ **do**
    ▷ Get important sensors for the activity
    $S_A \leftarrow \Pi_{\mathcal{S}}[A]$
    ▷ Initialize a map for the current activity's rates
    $R_{map,A} \leftarrow$ new empty Map()
    for each sensor $s \in S_A$ **do**
        ▷ Find max frequency and calculate Nyquist rate
        $f_{max} \leftarrow \text{FFT}(\text{Signal}(\mathcal{D}, A, s, \theta_{FFT}))$
        $R_{map,A}[s] \leftarrow 2 \times f_{max}$
    $\Pi_{\mathcal{R}}[A] \leftarrow R_{map,A}$
return $\Pi_{\mathcal{R}}$

For a specific activity, the signal data from each essential sensor is transformed into the frequency domain using a Fast Fourier Transform (FFT). The analysis then identifies the maximum frequency component needed to capture a predefined portion of the signal's total energy, as determined by an energy threshold (θ_{FFT}). According to

the Nyquist-Shannon theorem, the minimum sampling rate required to reconstruct the signal without losing this information is twice this maximum frequency. The strategy then selects the lowest available discrete sampling rate from a predefined list (e.g., [1, 2, 5, 10, 25, 50] Hz) that meets or exceeds this calculated minimum ($R_{map,A}[s]$). This per-sensor optimization results in an efficient policy ($\Pi_{\mathcal{R}}$) where, for the same activity, a high-frequency sensor like an accelerometer may be sampled at 50 Hz while a slower-changing sensor like a magnetometer is sampled at only 10 Hz, significantly reducing overall data volume and energy consumption. The tuning of the FFT energy threshold (θ_{FFT}) allows for a trade-off between data quality and energy consumption.

5.5.3 Real time application of Sensor Selection and Sampling

During real-time operation, the system combines knowledge from its offline analysis with live data to make dynamic sensing decisions (Algorithm 7). The wearable model periodically infers the user's current activity (A_{pred}) based on a window of sensor data. This inference frequency is governed by the Decision Interval ($T_{interval}$), a tunable parameter balancing responsiveness and computation. An activity is confirmed if its prediction confidence exceeds a strict confidence threshold (θ_{conf}).

The system maintains a short-term history (H) of these predictions to assess the stability of the current activity. If the last few predictions are consistent, the system assumes a stable activity and enforces the specific pre-computed policy ($\Pi_{\mathcal{S}}, \Pi_{\mathcal{R}}$) for that activity. If the system is unstable and multiple activities are detected at the moment, then we use a fallback confidence threshold (β_{conf}) to select a subset of activities and the policies corresponding to those activities are enforced.

Finally, if the system in unstable and no activity predictions are above β_{conf}, the system enters a high-fidelity safe mode (C_{safe}). In this mode, it activates a default set of sensors at an average sample rate to ensure the new activity is accurately identified. This default set is determined beforehand by applying PFI to the entire training dataset. Conversely, once predictions stabilize for a predefined duration, the system consults the pre-computed policy map. It then retrieves and applies the specific set of essential sensors ($\Pi_{\mathcal{S}}$) and their optimized sampling rates ($\Pi_{\mathcal{R}}$) corresponding to the stable activity. This policy remains active until the next transition is detected. This dual-mode approach ensures maximum data quality during periods of uncertainty and

maximum energy efficiency during periods of stability.

Algorithm 7 Real-Time Adaptive Sensing

Input: Policies $\Pi_{\mathcal{S}}$, $\Pi_{\mathcal{R}}$; Thresholds $\theta_{conf}, \beta_{conf}$; Safe Config C_{safe}; Wearable model M_w
▷ Initialization
$H \leftarrow$ new Queue() ▷ History buffer
$C_{current} \leftarrow C_{safe}$
while true **do**
    ▷ Wait for next Decision Interval
    Wait($T_{interval}$)
    $D_{window} \leftarrow$ CollectData($C_{current}$)
    $probs \leftarrow M_w$.Predict(D_{window})
    $A_{top} \leftarrow$ argmax($probs$)
    ▷ Update History
    H.Enqueue(A_{top})
    if IsStable(H) **and** $probs[A_{top}] \geq \theta_{conf}$ **then**
        ▷ Stable State: Apply specific optimal policy
        $C_{current} \leftarrow$ Config($\Pi_{\mathcal{S}}[A_{top}], \Pi_{\mathcal{R}}[A_{top}]$)
    else
        ▷ Unstable: Check for Fallback Candidates
        $Candidates \leftarrow \{c \in \mathcal{A} \mid probs[c] \geq \beta_{conf}\}$
        if $Candidates \neq \emptyset$ **then**
            ▷ Fallback State: Union of likely policies
            $S_{union} \leftarrow \bigcup_{c \in Candidates} \Pi_{\mathcal{S}}[c]$
            $R_{union} \leftarrow$ AverageRateMap($Candidates, \Pi_{\mathcal{R}}$)
            $C_{current} \leftarrow$ Config(S_{union}, R_{union})
        else
            ▷ Unknown State: Revert to Safe Mode
            $C_{current} \leftarrow C_{safe}$
    Apply($C_{current}$)

5.5.4 Server-Side Inference

Data collected by the wearable according to the dynamically adapted policy is periodically transferred to an edge server for final, high-accuracy activity inference using the more powerful edge model (Figure 5.12). This data transmission, orchestrated via Bluetooth Low Energy (BLE), can be triggered at regular intervals or when the on-device memory buffer approaches its capacity. The energy-efficient sensing strategies offer a dual benefit: in addition to reducing sensor power consumption, they also decrease the data payload that must be transmitted, which in turn lowers the total energy cost of communication. Upon receiving the sparsely-sampled data, the edge server first performs a reconstruction step, upsampling the signal from each sensor back to the original base sampling rate via signal interpolation. This reconstructed data is then fed into the edge model for final activity classification.

5.6 Evaluation

5.6.1 Experiment Setup

This section details the methodology used to evaluate Viveka. The experiments are designed to answer the following questions:

- How significant are the energy savings achieved by our method compared to state-of-the-art approaches?
- What is the trade-off between the achieved energy savings and the resulting HAR performance?

Implementation Details: The simulation framework is developed in Python. We utilized the scikit-learn library for data preprocessing and employed TensorFlow and TensorFlow Lite (TFLite) to train and optimize models for on-device inference on resource-constrained hardware. All models were trained on a desktop system running Ubuntu 24.04.3 LTS, equipped with a 13th Gen Intel Core i7-13700HX processor, 16 GB of RAM, and an NVIDIA GeForce RTX 4060 GPU.

Dataset: Our evaluation uses two real world datasets (Table 5.1):

- MHEALTH [132, 133]: It comprises recordings from 10 volunteers performing 12 physical activities and features data from inertial sensors (accelerometer, gyroscope, magnetometer) placed on the subject's chest, right wrist and left ankle. A two-lead ECG sensor was also placed on the chest. The dataset contains data from eight sensors in total, all of which were sampled at 50 Hz.
- PAMAP2 [134]: It consists of recordings from 9 volunteers performing 18 activities wearing sensors on hand, chest and ankle (accelerometer, gyroscope, magnetometer) and a heat rate monitor. There are 13 sensor features in the dataset sampled at 100Hz.

Table 5.1: Multi-model sensor datasets

Dataset	Users	Sensor Placement	Sensors (Axes)	Rate
MHEALTH	10	Chest, R. Wrist, L. Ankle	Acc, Gyro, Mag, ECG (8)	50 Hz
PAMAP2	9	Hand, Chest, Ankle	Acc, Gyro, Mag, HR (13)	100 Hz

HAR Models and Training Protocol: We employed the TinierHAR model [135] for our experiments. For all datasets, sensor data was segmented using a 4-second sliding window with a 50% overlap. The dataset was partitioned into training (80%) and testing (20%) sets using a temporal split to ensure the model's ability to generalize to new, unseen data. The model on the smart wearable is a lightweight version of the more complex TinierHAR model executed on the edge server. On the wearable, inference is performed periodically based on a decision interval and the impact of this execution frequency on performance and energy is systematically evaluated in our results.

5.6.2 Baseline

To rigorously evaluate our proposed method, Viveka, we compare it against multiple baselines that span a wide spectrum of approaches. These baselines are systematically designed to represent the state-of-the-art and to isolate the individual contributions of our system's core components: sensor selection (global vs. activity-aware) and sampling rate adaptation (static vs. dynamic).

1. **Oracle**: The Oracle is assumed to have perfect, a priori knowledge of the user's true activity at all times. For any given activity, it activates only the optimal,

predetermined set of sensors and operates them at the minimum sampling rate required, as determined by the Nyquist-Shannon theorem. This baseline is not a practical competitor but serves to contextualize performance by establishing the best achievable result.

2. **All Sensors On (AllSensorsOn)**: This baseline represents a standard, non-adaptive approach. All available sensors are continuously active, operating at the maximum static sampling frequency. It serves as an upper bound for data sensing and energy consumption.

3. **Variance-based Sampling (VS)**: In this baseline, all sensors remain active, but their sampling rates are adapted based on signal variance similar to [121]. The rate is increased during periods of high variance (indicating activity changes) and decreased during periods of low variance.

4. **Global Sensor Selection (GS)**: Only the globally most important subset of sensors, identified by PFI, is activated. This fixed set of sensors operates at the maximum static sampling rate throughout the evaluation which mimics [113,115].

5. **Global Sensor Selection with Variance-based Sampling (GS-VS)**: This baseline combines global selection with variance-based sampling. The globally important sensor subset is always active, but its sampling rates are dynamically adjusted based on signal variance [113, 136].

6. **Activity-based Sensor Selection (AS)**: Based on the inferred activity, a specific subset of sensors is activated. All sensors in this active set operate at a fixed, maximum sampling rate. It is a modified version of [124] with fixed sampling rates.

7. **Activity-based Sampling Rate (AR)**: In this approach, all sensors are active. Upon inferring an activity change, the sampling rates for all sensors are increased uniformly. When the activity stabilizes, the rates are gradually decreased [136].

8. **Activity-aware Sensor Selection with Variance-based Sampling (AS-VS)**: This approach activates an activity-specific set of sensors and then further

optimizes power by adapting their sampling rates uniformly based on real-time signal variance. It is a modified version of [124] with sampling rate variation.

9. **Viveka**: Our proposed approach, where sensors are selected based on the inferred activity and their sampling rates are individually adjusted according to the spectral energy analysis.

Table 5.2: Sensors for experiment

Sensor type	Commercial sensor
Accelerometer	BMI160 [137]
Gyroscope	LSM6DSO [138]
Magnetometer	BMM350 [139]
ECG	MAX30003 [140]
Temperature	NST112 [141]
BLE	nRF52840 [142]

5.6.3 Evaluation Metrics

We analyze the performance of the proposed method against the baselines using metrics for classification (F1-score), data overhead (data reduction, normalized data collection) and energy consumption (energy savings, normalized energy consumption).

- **Classification Performance**: The primary metric for HAR performance is the F1-score, which is the harmonic mean of precision and recall. Precision measures the accuracy of positive predictions, while recall measures the ability to capture all actual positive instances. We chose the F1-score due to its robustness in handling the class imbalances often present in real-world activity datasets. It is calculated as:

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.8}$$

  We compare the F1-score of our method with the baselines to quantify any impact on classification accuracy resulting from our data reduction techniques.

- **Data Reduction**: We quantify data reduction by comparing the total volume of data collected by our approach to that of the All Sensors On baseline. The result is then presented as a normalized value relative to the baseline.

- **Energy Consumption**: We use an energy model that provides a holistic estimate of power consumption by aggregating three key components: sensing, computation, and communication. The total energy, E_{total}, is defined as:

$$E_{\text{total}} = E_{\text{sense}} + E_{\text{compute}} + E_{\text{comm}} \tag{5.9}$$

  Each component is modeled as follows:

  - **Sensing Energy (E_{sense})**: This component is calculated using power consumption values from the datasheets of commercially available sensors, as detailed in Table 5.2. Our model accounts not only for the energy consumed during active sampling but also for the transactional energy costs of state transitions (i.e., enabling/disabling sensors and changing their sampling rates).
  - **Computation Energy (E_{compute})**: The energy cost of a single HAR model inference is measured empirically. We executed the model for 10,000 consecutive iterations on our target smartwatch (TicWatch Pro 5) and measured the average energy draw using the Android Power Profiler. The energy consumptions are $\approx$ 15.2 mJ and $\approx$ 47.2 mJ per inference for MHEALTH and PAMAP2 respectively.
  - **Communication Energy (E_{comm})**: The energy required for data transmission is modeled based on specifications from the datasheet for the Nordic Semiconductor nRF52 series Bluetooth Low Energy (BLE) chipset, a common component in modern wearables.

5.6.4 Overall Impact of Viveka

In this evaluation, we compare the proposed Viveka framework against the state-of-the-art sensor selection strategies (Figure 5.13). The results shown in Figure 5.13 corresponding to the best parameter values { PFI threshold, FFT Energy threshold,

Decision interval, Prediction confidence threshold, Fallback confidence threshold} for Viveka: MHEALTH {0.005, 0.80, 2s, 0.80, 0.25} and PAMAP2 {0.01, 0.99, 4s, 0.80, 0.25}.

For MHEALTH, Viveka attains an F1-score of 0.86, which is equivalent to the high-accuracy static baselines, while simultaneously reducing energy consumption by 70.2% and data generation by 77.8%. In PAMAP2, despite the dataset's complexity, Viveka achieves an F1-score of 0.65 while achieving 63.5% energy savings and 75.0% data reduction.

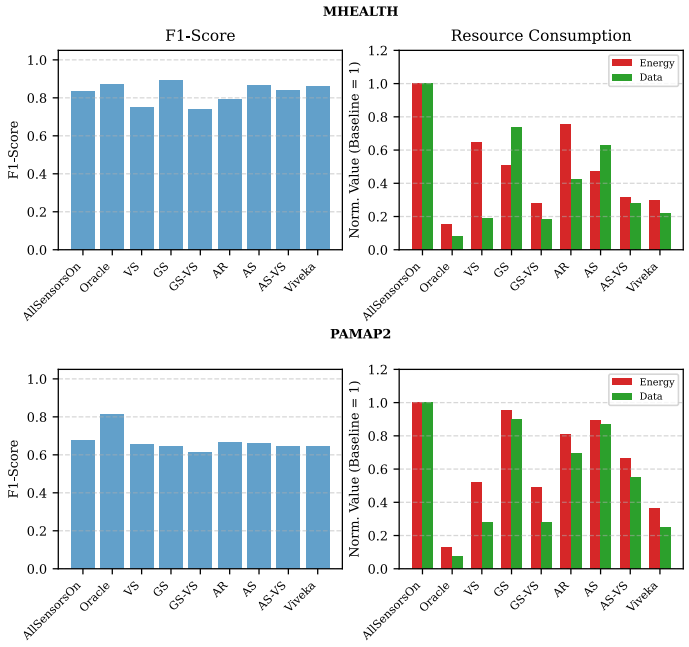


Figure 5.13: Overall performance comparison of Viveka against SOTA approaches on MHEALTH (top) and PAMAP2 (bottom). The left column reports F1-Score, while the right column displays normalized energy and data consumption (Lower is better). Viveka achieves the most favorable trade-off, delivering accuracy comparable to static methods with significantly lower resource costs ($< 40\%$ of baseline).

While AS performs well on accuracy (F1=0.86 for MHEALTH), it is inefficient. For PAMAP2, AS consumes 90% of the baseline energy, whereas Viveka achieves comparable accuracy (F1=0.65 vs. 0.66) using only 36% of the energy. Naive dynamic strategies often sacrifice accuracy for savings. In PAMAP2, GS-VS drops to an F1-score of 0.61.

Viveka outperforms it in both accuracy (+4% F1) and efficiency (12% lower energy consumption), demonstrating smarter sensor selection.

Takeaway: The Viveka framework successfully decouples sensing cost from classification accuracy. Unlike static strategies that over provision resources (wasting energy) or naive dynamic strategies that under provision (hurting accuracy), Viveka dynamically identifies the minimal sufficient sensor set. This results in a system that matches the fidelity of energy hungry static baselines while exceeding the efficiency of aggressive dynamic approaches.

5.6.5 Parameter Sensitivity Analysis

To evaluate the robustness of Viveka, we performed a sensitivity analysis for each hyperparameter. The resulting impact on system performance is measured as the deviation between the best and worst observed outcomes and is illustrated in Figure 5.14.

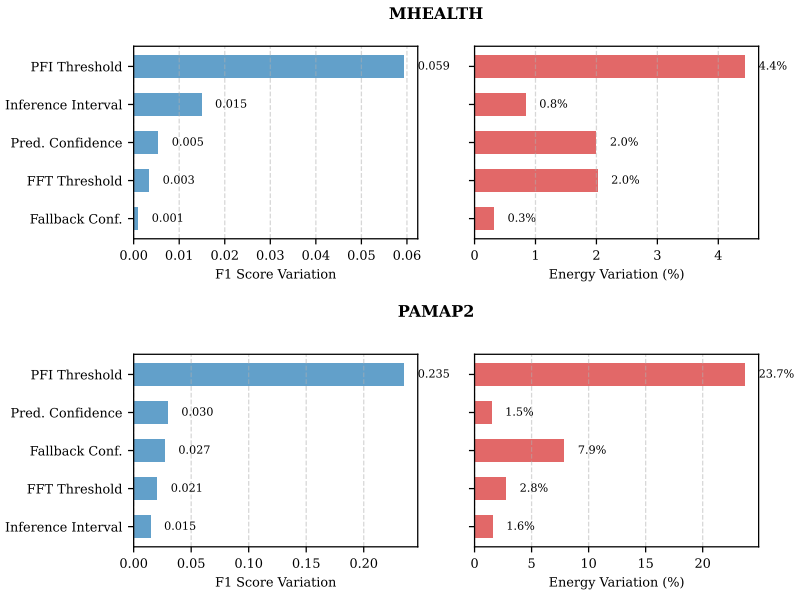


Figure 5.14: Parameter sensitivity analysis. The results highlight PFI theshold and FFT Energy Threshold as the dominant parameters. Theother parameters are dependent onthe stability of activities in the dataset.

Architectural Parameters: The **PFI Threshold (θ_{PFI})** and **FFT Energy Threshold (θ_{FFT})** consistently exhibit the widest performance variance across both datasets.

These parameters function as the system's architectural constraints as they define the active set of sensors and their sampling rate. The misconfiguration of these parameters could lead to the most significant penalties. For instance, setting θ_{PFI} too aggressively removes critical sensors (like the gyroscope in dynamic activities), causing an immediate collapse in F1-score. Conversely, setting it too loosely negates energy savings by keeping redundant sensors active.

Runtime Parameters: The runtime parameters **Decision Interval ($T_{interval}$)**, **Prediction Confidence (θ_{conf})**, and **Fallback Threshold (β_{conf})**, govern the system's dynamic behavior. Their sensitivity is inversely related to domain stability.

Based on the sensitivity analysis, we propose a hierarchical tuning strategy for deploying Viveka. This approach moves from defining hardware feasibility to optimizing runtime behavior. First, we establish the minimal hardware baseline by calibrating PFI Threshold (θ_{PFI}) and FFT Energy Threshold (θ_{FFT}) to determine the minimal set of sensors and their minimum sampling rate required. Second, we tune the Decision Interval ($T_{interval}$) and Prediction Confidence (θ_{conf}) to determine the optimal trade-off between energy savings and accuracy. Finally, we calibrate the Fallback Threshold (β_{conf}) to ensure the system is robust to misclassifications.

5.6.6 PFI Threshold Tuning

The Permutation Feature Importance (PFI) threshold (θ_{PFI}) acts as the primary gatekeeper for sensor selection. In this experiment, we compare the impact of PFI threshold tuning across the two datasets. The parameter values {FFT Energy threshold, Decision interval, Prediction confidence threshold, Fallback confidence threshold} for Viveka: MHEALTH {0.80, 2s, 0.80, 0.25} and PAMAP2 {0.99, 4s, 0.80, 0.25}.

Figure 5.15 show a clear inverse relationship: as the threshold increases, the system selects fewer sensors, increasing energy savings but risking accuracy loss. For MHEALTH, the accuracy remains stable (F1 $\approx$ 0.86) even with aggressive pruning of threshold up to 0.01. As for PAMAP2, the accuracy degrades sharply beyond a very conservative threshold (> 0.005), indicating that even weak features are critical for distinguishing complex activities.

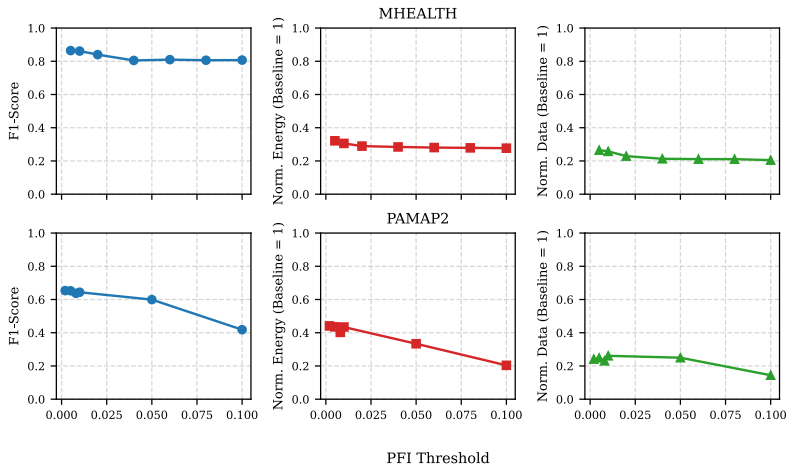


Figure 5.15: Impact of the Permutation Feature Importance (PFI) threshold. Increasing the threshold aggressively prunes sensors, improving efficiency (lower energy/data) but degrading accuracy. For MHEALTH, 0.01 offers the optimal trade-off, whereas PAMAP2 requires a threshold of 0.005 to maintain classification fidelity.

Takeaway:The PFI threshold effectively maps to the rigor of the sensor signals within the dataset. A low-rigor system (e.g., MHEALTH) requires a significantly lower number of sensors per activity to distinguish patterns, allowing for a higher threshold and more aggressive energy savings. Conversely, a high-rigor system (e.g., PAMAP2) relies on intricate inter-sensor correlations, necessitating a lower threshold to retain a higher density of sensors for accurate classification.

5.6.7 FFT Energy Threshold Tuning

This parameter controls how much spectral information is preserved during feature extraction. In this experiment, we compare the impact of FFT energy threshold (θ_{FFT}) tuning across the two datasets. The parameter values {PFI Threshold, Decision interval, Prediction confidence threshold, Fallback confidence threshold} for Viveka: MHEALTH {0.01, 2s, 0.80, 0.25} and PAMAP2 {0.005, 4s, 0.80, 0.25}.

In MHEALTH, the F1-score is virtually flat across all thresholds, suggesting the activities (e.g., walking, jogging) are distinct enough in the time domain that even partial spectral energy (80%) is sufficient for classification. As for PAMAP2, the accuracy improves with the threshold, peaking at 0.99. This indicates that high-intensity, complex

activities require high-fidelity frequency data to be distinguished from noise.

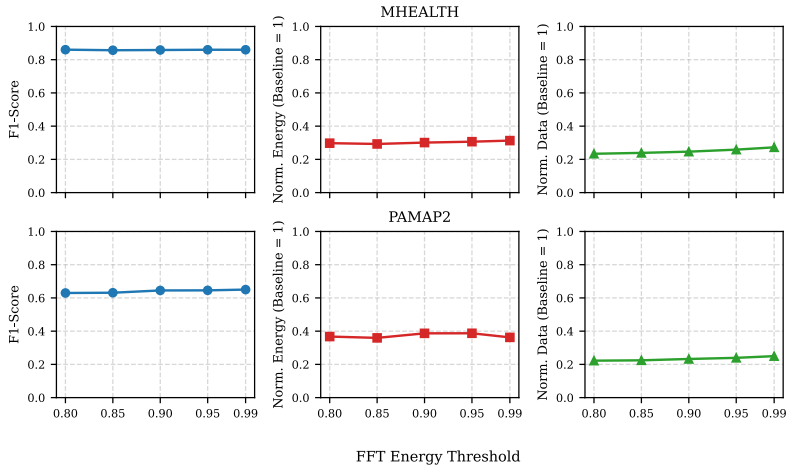


Figure 5.16: Impact of FFT energy threshold. MHEALTH shows stable performance across all thresholds, indicating distinct spectral features. In contrast, PAMAP2 benefits from a higher threshold of 0.99, as retaining more spectral energy is necessary to accurately classify its complex, high-intensity activities without significant energy penalty.

Takeaway:Retaining high spectral energy (99%) is important for complex datasets like PAMAP2. However, for stable activity datasets such MHEALTH, the spectral enery retained could be lower, with minimal impact on classication accuracy.

5.6.8 Impact of Decision Interval

This parameter dictates how often the system wakes up to evaluate the current activity. In this experiment, we compare the impact of inference decision interval ($T_{interval}$) tuning across the two datasets. The parameter values {PFI Threshold, FFT Energy threshold, Prediction confidence threshold, Fallback confidence threshold} for Viveka: MHEALTH {0.01, 0.80, 0.80, 0.25} and PAMAP2 {0.005, 0.99, 0.80, 0.25}.

Increasing the interval from 2s to 16s reduces the number of inferences, lowering the computational load on the wearable. For MHEALTH, it favors faster reactions (2s), while PAMAP2 actually benefits from the smoothing effect of a slightly longer window (4s), filtering out subtle movements that might confuse the classifier.

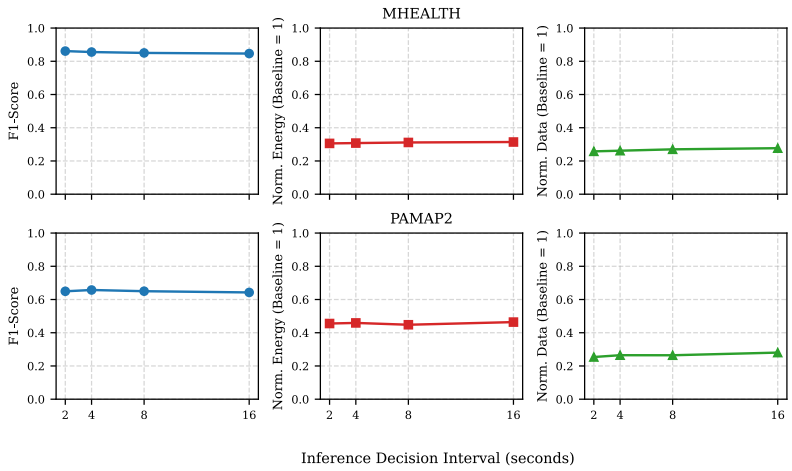


Figure 5.17: Impact of Decision interval. Shorter intervals (2s) maximize responsiveness for MHEALTH, while PAMAP2 achieves peak accuracy at 4s. While longer intervals (8s, 16s) degrade F1-score slightly due to delayed reaction times, they offer reductions in computational load and sensor usage.

Takeaway: Minimizing the decision interval is critical for energy efficiency. A shorter interval (e.g., $2s - 4s$) ensures the sensor configuration adapts to user behavior quickly. The extension of the interval could result in a high-power state (due to delayed detection) exceeding the computational savings of running fewer inferences.

5.6.9 Confidence thresold tuning

This threshold (θ_{conf}) determines when the system trusts its current lightweight sensor set versus triggering a fallback. It is applicable to the predictions of the light weight model running on the wearable. The parameters values {PFI Threshold, FFT Energy threshold, Decision interval, Fallback confidence threshold} for Viveka in this experiment for the datasets are: MHEALTH {0.01, 0.80, 2s, 0.25} and PAMAP2 {0.005, 0.99, 4s, 0.25}.

Increasing the threshold ($0.6 \rightarrow 0.8$) improves F1-score by rejecting uncertain predictions. For threshold of 1.0, it forces the system to trigger fallback mechanisms too frequently, hurting energy savings without providing a statistically significant gain in accuracy compared to the threshold 0.8.

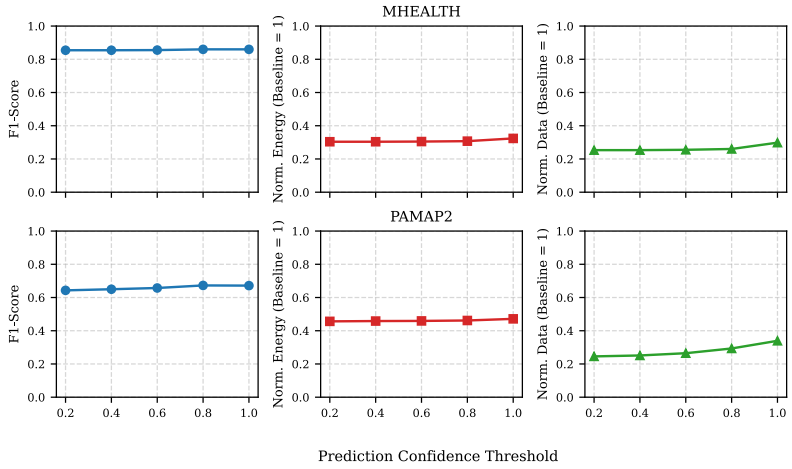


Figure 5.18: Impact of Confidence threshold. Increasing the confidence requirement (up to 0.8) generally improves F1-score by preventing the system from acting on uncertain predictions. However, strict thresholds (1.0) reduce energy savings, as the system is forced to trigger fallback mechanisms or activate more sensors more frequently.

Takeaway: Regardless of dataset complexity, a prediction confidence threshold of 0.80 provides the best tradeoff between accuracy and performance. It is sufficien to filter out low-confidence errors (noise), yet lenient enough to prevent the system from becoming overly conservative. Increasing the threshold (e.g., to 1.0) forces the system to trigger energy expensive fallback mechanisms too frequently without much improvement in accuracy.

5.6.10 Fallback threshold tuning

This parameter (β_{conf}) controls how selective the system is when it attempts to recover from a low-confidence state using more sensors. The parameters values {PFI Threshold, FFT Energy threshold, Decision interval, Confidence thresold} for Viveka in this experiment for the datasets are: MHEALTH {0.01, 0.80, 2s, 0.80} and PAMAP2 {0.005, 0.99, 4s, 0.80}.

The graphs show that a lower threshold ($\approx 0.15 - 0.25$) yields the highest accuracy. For MHEALTH, given the model is detecting activities with high accuracy, there is less fallbacks. Setting the threshold too high (> 0.35) causes a drop in F1-score for PAMAP2. This implies that if the system rejects a fallback prediction, it often has no

better alternative, leading to unclassified segments.

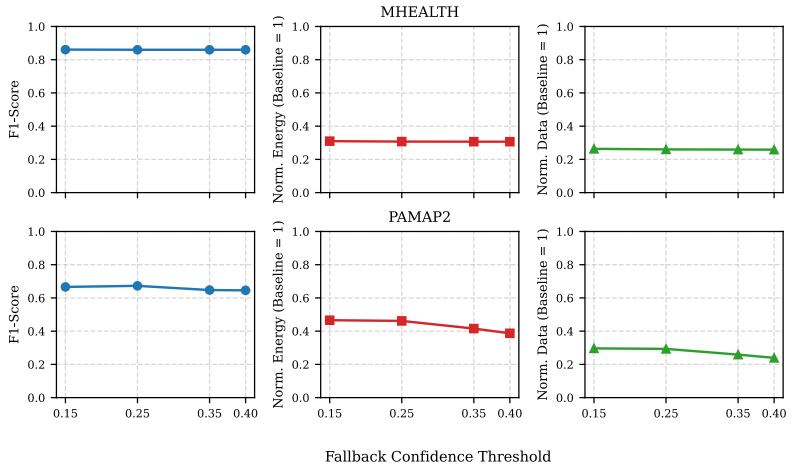


Figure 5.19: Impact of the Fallback Confidence Threshold. A lower threshold (e.g., 0.25) allows the system to successfully classify activities using a subset of sensors during fallback. Excessively high thresholds (> 0.35) results in a drop in F1-score due to incorrect fallback predictions.

Takeaway: The recovery mechanism in Viveka falls towards lineancy over strict efficiency. If the current sensor data points to multiple activities, then sensors corresponding to those sensors are activated to identify the correct activity. A moderate threshold of 0.25 ensures the system captures the activity for the two datasets.

5.6.11 Sensor Activations

In this analysis, we examine the granular sensor activations of the MHEALTH dataset, configured with the optimal parameters identified in the previous sections. The sensor activation map (Figure 5.20) provides a granular view of the Viveka framework's runtime behavior, confirming its ability to decouple sensing cost from continuous monitoring.

The transitions in the sensor heatmap (bottom) align perfectly with the activity transitions in the ground truth ribbon (top). Even Transient periods (brief transitions between activities) are captured with distinct sensor configurations. The system does not merely classify steady states, it is highly responsive to state changes, correctly identifying the start and end of specific movements without significant lag.

The rows corresponding to Gyroscopes remain predominantly white (0 Hz) or light green (1 Hz) across most activities. They are only activated (darker colors) during specific complex tasks. This confirms that the most power-hungry sensors are often redundant for recognizing common human activities. The system successfully prunes them, relying instead on cheaper modalities (Accelerometers), thereby validating the PFI feature selection strategy.

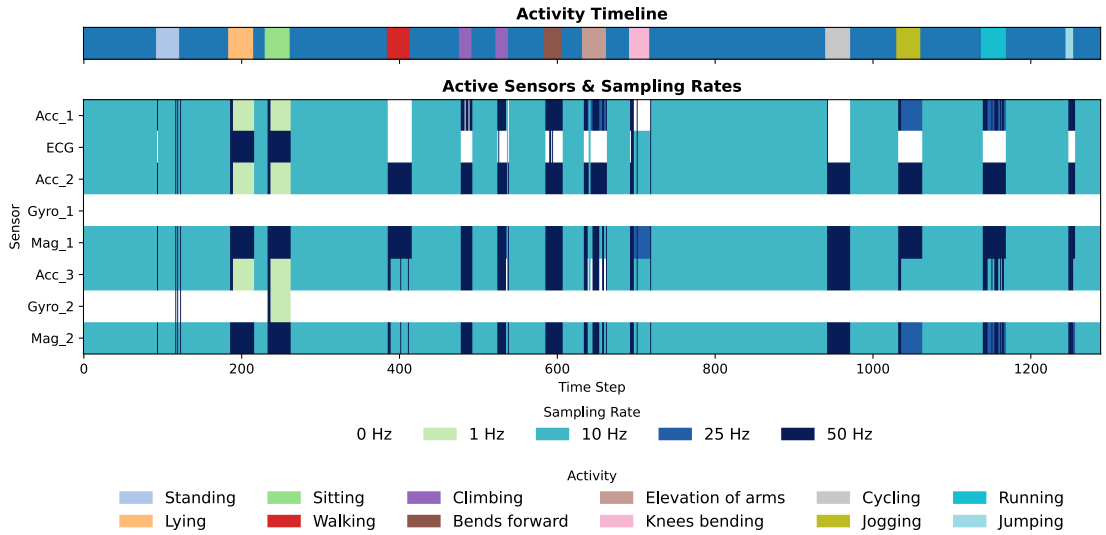


Figure 5.20: Sensor activations for MHEALTH. The top ribbon displays the ground truth activity timeline, while the heatmap below illustrates the dynamic sensor selection and sampling rate adaptation (0-50 Hz) triggered by the Viveka framework. It highlights the system's ability to selectively prune sensors and scale sampling rate according to activity intensity.

There is a clear correlation between activity intensity and color saturation. High-intensity activities (e.g., Jogging, Running) trigger dark navy blocks (50 Hz), while low-intensity activities (e.g., Sitting, Standing) trigger light green/teal blocks (1-10 Hz). Viveka exhibits context-aware scalability. It dynamically scales the sampling rates proportional to the activity rigor, ensuring no energy is wasted oversampling stationary behavior.

The vertical slices of the heatmap show that for any given time step, only a fraction of the available sensors are active (non-white). Very rarely is the entire column lit up. Viveka achieves sparsity, proving that a full sensor suite is almost never required

simultaneously. Each activity requires only a minimal subset of sensors.

Within continuous activity blocks, the color patterns remain consistent (solid blocks of color rather than flickering noise). The strategy demonstrates stability. It avoids sensor thrashing (rapidly toggling sensors On/Off), which is critical because the energy overhead of powering up a sensor often outweighs the savings of turning it off for a split second.

Takeaway: The visualization confirms that Viveka achieves efficiency through context-aware semantic adaptation, replacing static sampling schedules with dynamic, stable and minimalist sensing signatures tailored to the instantaneous activity rigor of the user.

5.6.12 Projected Impact on Real-World Hardware

To translate our experimental energy savings into user benefits, we model the impact of Viveka on the battery life of two common wearable types: a Smartwatch and a Smart Ring. We utilize power consumption breakdowns from recent component-level studies [143, 144] to estimate the theoretical lifespan extension.

Smartwatch: Marathon Scenario

Standard smartwatches are typically constrained by display power. However, during long duration activity tracking (e.g., marathons), the display is inactive and the energy profile shifts dramatically. According to [143], while the display dominates interactive use, the CPU and sensor subsystem account for over 60% of the power drain during background operations. We assume a baseline battery life of 18 hours (typical for flagship smartwatches). In a Screen-Off/Tracking-On state, sensing and inference consume $\approx 70\%$ of the total power budget. Viveka reduces this specific sensing load by 70.2%. This reduction lowers the total system power consumption by $\approx 49\%$, theoretically extending the single-charge tracking duration from **18 hours to 35.4 hours**. This implies that Viveka could enable high-fidelity smartwatches to track multi-day activities without requiring a recharge.

Smart Ring: Monitoring Scenario

Smart rings represent the ideal use case for Viveka, as they lack a power hungry display and rely entirely on sensing efficiency. [144] analyze the current consumption of ring-embedded optical sensors, demonstrating that the Analog Front End (AFE) and LED drivers for PPG/IMU sensing constitute the dominant load ($\approx$ 80%), far exceeding the Bluetooth Low Energy (BLE) transmission cost when data is batched. Applying Viveka's context-aware pruning reduces the aggregate power draw significantly. For a smart ring with a baseline **5 day** battery life, Viveka projects an extension to $\approx$ **11.2 days**. This doubling of lifespan addresses a critical usability pain point, potentially allowing for nearly two weeks of operation on a tiny 15-20 mAh battery.

Table 5.3: Projected Battery Life Extension

Device Type	**Dominant Load**	**Sensing Share**	**Baseline Life**	**Viveka Life**
Smartwatch	Sensors + CPU	$\sim$70%	18 Hours	**35 Hours**
Smart Ring	Optical/IMU AFE	$\sim$80%	5 Days	**11 Days**

5.7 Related Work

Energy efficiency is a paramount concern in wearable Human Activity Recognition (HAR), driving research into resource-aware methodologies. The literature has largely pursued this goal through two primary avenues: sensor selection and adaptive sampling, with the most advanced works combining the two.

Sensor Selection Strategies: Sensor selection methods aim to reduce power consumption by activating only the most informative sensors. Static approaches pre-determine an optimal sensor set using techniques like game theory [115], bio-inspired algorithms [116], or cost optimization [113]. However, their fixed nature limits adaptability to dynamic user states. Dynamic methods offer more flexibility by using Bayesian models to quantify uncertainty (VFDS) [117] or employing specialized architectures that can internally weigh sensor importance (DANA) [118]. The primary drawback of these advanced techniques is that they often require complex, model-specific architectures, sacrificing the modularity needed to work with general-purpose HAR models.

Adaptive Sampling Techniques: Adaptive sampling techniques focus on reducing data volume by modulating the sampling frequency. These can be context-aware [119], using a lightweight model to infer the activity and adjust rates accordingly (FreqSense) [120], or purely data-driven, reacting directly to signal properties like variance (LASA-IoT) [121]. A key limitation is that they often apply a uniform sampling rate to all active sensors, failing to account for the fact that different sensors have different information content and power costs, leading to suboptimal energy savings.

Co-optimization of Selection and Sampling: Recognizing that these problems are coupled, the most advanced research seeks to co-optimize both selection and sampling. A prevalent paradigm is the hierarchical system, where a low-power configuration detects a potential change, triggering a more resource-intensive mode for precise identification [145]. While effective, these systems often act as a simple binary switch, lacking the granularity to modulate individual sampling rates or select specific sensor subsets during transitions. More sophisticated frameworks like CoSS [125] use reinforcement learning to learn a joint policy. However, this approach is model-integrated, requiring specialized training and resulting in a static configuration that cannot adapt to real-time dynamics post-deployment.

5.8 Conclusion

The full potential of smart wearables hinges on extending their battery life, as frequent recharging creates data gaps and user inconvenience. To address this, we presented Viveka, a context-aware sensing framework designed to dramatically improve both energy and data efficiency. Viveka adapts to the user's context by using permutation-based feature importance to select the most relevant sensors and the spectral energy analysis to determine their optimal sampling rates for the current activity. Our framework also includes robust fallback mechanisms to ensure high accuracy is maintained, even when on-device context detection is uncertain. Our extensive evaluation on the MHEALTH and PAMAP2 datasets demonstrates that Viveka achieves the best tradeoff between accuracy and energy efficiency compared to existing strategies. The framework achieves 63-70% energy savings and 75-78% data reduction compared to baseline systems, while maintaining F1-scores within 3-5% of the theoretical Oracle. We also conducted a sensitivity analysis across all the parameters used in the framework to identify the sweet

spot and an order of tuning.

5.8.1 Chapter Takeaways

⇒ *Co-optimization is Key*: Treating sensor selection and adaptive sampling independently leads to suboptimal results. Dynamically co-optimizing both based on activity context maximizes energy and data efficiency in wearables.

⇒ *Activity-Specific Sensing Policies are Effective*: Different activities have distinct sensor and sampling rate requirements. Using techniques like PFI to identify activity-specific sensor needs and Nyquist-Shannon to determine minimum required sampling rates allows for significant resource savings.

⇒ *Robustness Requires Fallbacks*: Dynamic adaptation needs mechanisms (like Viveka's "safe mode") to handle uncertainty during activity transitions or low-confidence predictions, ensuring accuracy is maintained even while optimizing for energy.

⇒ *Sensing Dominates Energy Consumption*: For HAR on wearables, the energy consumed by active sensors vastly outweighs computation and communication costs. Effective energy-saving strategies must prioritize reducing sensor activity and data rates. Viveka achieves substantial savings (up to 62%) by directly targeting this dominant factor.

Chapter 6

Concluding and Future Work

6.1 Summary of Contributions

Data management at the network's edge presents significant challenges related to resource constraints, heterogeneity, dynamic conditions and stringent application requirements. Effectively realizing the full potential of edge computing hinges on developing intelligent strategies to handle this data deluge efficiently. This thesis argued for and demonstrated the efficacy of adopting application awareness as a core principle to tackle these challenges. We developed and evaluated four distinct techniques across the data lifecycle, targeting different application scenarios, to improve the performance, efficiency, and responsiveness of edge systems.

The contributions provide a multi-faceted solution:

1. First, we addressed the need for persistent storage in volatile edge environments by designing Cargo, the storage layer for the Armada framework (Chapter 2). Cargo demonstrated the ability to provide reliable, low-latency data access through compute-proximity based replica placement and seamless failover. It proved significantly more effective than relying solely on cloud backups, especially when leveraging volunteer resources.

2. Second, focusing on optimal data placement, we compared distance, latency, and

spatial-awareness strategies (Chapter 3). Our findings revealed that while latency-based selection is effective, the spatial heuristic offers superior scalability by efficiently pruning the search space, making it ideal for dense edge deployments without sacrificing significant performance.

3 Third, for demanding applications like Mobile Augmented Reality, we developed ASTRA, an application-aware prefetching framework (Chapter 4). By integrating object associations, temporal relevance, and spatial proximity, ASTRA significantly improved cache hit rates (up to 35%) and reduced end-to-end latency (up to 14%) compared to baseline methods, with an adaptive tuning mechanism providing further gains.

4 Finally, optimizing at the data source, we proposed Viveka, an activity-aware co-optimization framework for smart wearables (Chapter 5). Viveka demonstrated dramatic energy savings (up to 62%) and data reduction (up to 74%) by dynamically selecting relevant sensors using PFI and determining optimal, per-sensor sampling rates via the Nyquist-Shannon theorem, all while maintaining high HAR accuracy.

Collectively, these contributions illustrate that applying application awareness systematically, from optimizing sensor sampling on end-devices (Viveka), to intelligently placing and prefetching data on edge servers (Data Placement Strategies, ASTRA), within a robust framework supporting stateful applications (Armada/Cargo), yields substantial improvements in latency, resource efficiency, and overall system performance. This work underscores that tailored, context-aware approaches are essential for unlocking the capabilities of heterogeneous edge computing infrastructures.

While this thesis provides significant advancements, certain limitations frame the scope of the work. The evaluations primarily relied on simulation and emulation, and real-world deployments at scale would introduce further complexities. The frameworks were tested with specific application types (AR, HAR, Face Recognition), and their performance may vary for other workload characteristics. Assumptions regarding network conditions and node behavior were made, and further investigation into highly unpredictable or adversarial environments is warranted.

6.2 Future Directions

Building upon the energy-saving techniques demonstrated in Viveka (Chapter 5), a primary future direction is enhancing on-device intelligence while minimizing power consumption. This involves developing lightweight, automated data management models using TinyML frameworks. Considering the stringent energy constraints of smart wearables and IoT devices, research is needed to design and implement highly efficient machine learning models capable of running directly on the device. Techniques like model pruning, quantization and adaptive execution will be crucial for automating local decisions regarding adaptive sensing, data filtering and potentially even computation offloading strategies. This will further extend device operational lifetime and reduce reliance on continuous connectivity, critical for applications like long-term chronic disease management and remote care.

A second major work extends the concepts of dynamic data placement and prefetching (Chapter 3, Chapter 4) towards a fully AI-Driven Autonomous Data Fabric. Current edge systems remain largely reactive to user movement and network changes. The vision is to create a proactive, self-organizing edge infrastructure that anticipates application needs and environmental dynamics. This necessitates research into AI and ML techniques for:

- Predictive Mobility and Hotspot Analysis: Developing models to predict user movement patterns and anticipate future high-density "hotspots" for specific applications.
- Proactive Data Management: Designing algorithms that use these predictions to proactively migrate data replicas, prefetch content (extending ASTRA's concepts), and reconfigure placements before performance degradation occurs, ensuring guaranteed low-latency service.
- Autonomous Resilience: Creating mechanisms for the data fabric to self-heal and re-route data access transparently in response to node failures or network partitions, enhancing reliability beyond simple replica switching. This autonomous

fabric would provide the resilient, ultra-low-latency backbone required for next-generation applications. These application include but not limited to truly collaborative AR/VR experiences across locations, real-time data sharing in autonomous vehicle networks for enhanced safety and coordination and robust public safety or disaster response systems where infrastructure may be compromised.

Finally, aligning with the growing need for sustainable computing, future work must address the energy consumption and environmental impact of the edge infrastructure itself. Given that a vast majority of data generation is shifting to the edge, it's critical to move beyond device-level efficiency towards system-level energy awareness. This includes:

- Energy-Aware Scheduling: Developing novel schedulers for edge and cloud tasks that explicitly consider dynamic energy costs, grid carbon intensity and the availability of renewable energy sources at edge locations.
- Optimizing Data Movement: Investigating data processing and transmission strategies specifically designed to minimize the energy footprint across the distributed edge-to-cloud continuum.
- Lifecycle Energy Analysis: Assessing the total energy cost, including embodied energy in hardware, for different edge deployment models. Success in this area can lead to more cost-effective edge/cloud services and reduce the environmental impact of large-scale IoT deployments in smart cities, agriculture and environmental monitoring.

References

[1] Wenxiao Zhang, Bo Han, and Pan Hui. Jaguar: Low latency mobile augmented reality with flexible tracking. In *Proceedings of the 26th ACM international conference on Multimedia*, pages 355–363, 2018.

[2] Santhosh Rao. What Edge Computing Means for Infrastructure and Operations Leaders. `https://www.gartner.com/smarterwithgartner/what-edge-computing-means-for-infrastructure-and-operations-leaders`, 2018. Accessed on 2025-05-04. Author attribution based on secondary source. Original publication date likely 2018.

[3] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4):14–23, 2009.

[4] Nour Alhuda Sulieman, Lorenzo Ricciardi Celsi, Wei Li, Albert Zomaya, and Massimo Villari. Edge-oriented computing: A survey on research and use cases. *Energies*, 15(2):452, 2022.

[5] Animesh Trivedi, Lin Wang, Henri Bal, and Alexandru Iosup. Sharing and caring of data at the edge. In *3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20)*, 2020.

[6] Simone Bolettieri, Raffaele Bruno, and Enzo Mingozzi. Application-aware resource allocation and data management for mec-assisted iot service providers. *Journal of Network and Computer Applications*, 181:103020, 2021.

[7] Martin Breitbach, Dominik Schäfer, Janick Edinger, and Christian Becker. Context-aware data and task placement in edge computing environments. In *2019 IEEE International Conference on Pervasive Computing and Communications (PerCom*, pages 1–10. IEEE, 2019.

[8] Mahadev Satyanarayanan, Guenter Klas, Marco Silva, and Simone Mangiante. The seminal role of edge-native applications. In *2019 IEEE International Conference on Edge Computing (EDGE)*, pages 33–40. IEEE, 2019.

[9] Junjue Wang, Ziqiang Feng, Shilpa George, Roger Iyengar, Padmanabhan Pillai, and Mahadev Satyanarayanan. Towards scalable edge-native applications. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, pages 152–165, 2019.

[10] Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. Real-time video analytics: The killer app for edge computing. *computer*, 50(10):58–67, 2017.

[11] Li Lin, Xiaofei Liao, Hai Jin, and Peng Li. Computation offloading toward edge computing. *Proceedings of the IEEE*, 107(8):1584–1607, 2019.

[12] Dimitrios Giouroukis, Alexander Dadiani, Jonas Traub, Steffen Zeuch, and Volker Markl. A survey of adaptive sampling and filtering algorithms for the internet of things. In *Proceedings of the 14th ACM international conference on distributed and event-based systems*, pages 27–38, 2020.

[13] Xin Qi, Matthew Keally, Gang Zhou, Yantao Li, and Zhen Ren. Adasense: Adapting sampling rates for activity recognition in body sensor networks. In *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 163–172. IEEE, 2013.

[14] Sushmita Ghosh, Swades De, Shouri Chatterjee, and Marius Portmann. Learning-based adaptive sensor selection framework for multi-sensing wsn. *IEEE Sensors Journal*, 21(12):13551–13563, 2021.

[15] Guangyu Yang, Lei Zhang, Can Bu, Shuaishuai Wang, Hao Wu, and Aiguo Song. Freqsense: Adaptive sampling rates for sensor-based human activity recognition

under tunable computational budgets. *IEEE Journal of Biomedical and Health Informatics*, 27(12):5791–5802, 2023.

[16] Abdulelah Alwabel. Latency-aware task offloading mechanism for mobile edge computing. *CLOUD COMPUTING 2025*, page 105, 2025.

[17] Lan Anh Nguyen, Sunggon Kim, and Yongseok Son. Edgeup: Utilization and priority-aware load balancing in edge computing. *Electronics*, 14(3):565, 2025.

[18] Xinliang Wei and Yu Wang. Popularity-based data placement with load balancing in edge computing. *IEEE transactions on cloud computing*, 11(1):397–411, 2021.

[19] Xinliang Wei, ABM Mohaimenur Rahman, and Yu Wang. Data placement strategies for data-intensive computing over edge clouds. In *2021 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, pages 1–8. IEEE, 2021.

[20] Ilija Hadžić, Yoshihisa Abe, and Hans Christian Woithe. Server placement and selection for edge computing in the epc. *IEEE Transactions on Services Computing*, 12(5):671–684, 2018.

[21] Tero Lähderanta, Teemu Leppänen, Leena Ruha, Lauri Lovén, Erkki Harjula, Mika Ylianttila, Jukka Riekki, and Mikko J Sillanpää. Edge computing server placement with capacitated location allocation. *Journal of Parallel and Distributed Computing*, 153:130–149, 2021.

[22] Xinglin Zhang, Zhenjiang Li, Chang Lai, and Junna Zhang. Joint edge server placement and service placement in mobile-edge computing. *IEEE Internet of Things Journal*, 9(13):11261–11274, 2021.

[23] Lei Huang. Armada: A robust latency-sensitive edge cloud in heterogeneous edge-dense environments. Master's thesis, University of Minnesota, 2021.

[24] Lei Huang, Zhiying Liang, Nikhil Sreekumar, Sumanth Kaushik, Abhishek Chandra, and Jon Weissman. Towards elasticity in heterogeneous edge-dense environments. In *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*, pages 403–413. IEEE, 2022.

[25] Nikhil Sreekumar, Abhishek Chandra, and Jon Weissman. Position paper: Towards a robust edge-native storage system. In *2020 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 285–292. IEEE, 2020.

[26] Kagami. go-face, 2021. Accessed on 2025-05-04.

[27] Apposite Technologies. Netropy emulator, 2021. Accessed on 2025-05-04.

[28] Gary B. Huang Erik Learned-Miller. Labeled faces in the wild: Updates and new reporting procedures. Technical Report UM-CS-2014-003, University of Massachusetts, Amherst, May 2014.

[29] Enzo Baccarelli, Paola G Vinueza Naranjo, Michele Scarpiniti, Mohammad Shojafar, and Jemal H Abawajy. Fog of everything: Energy-efficient networked computing architectures, research challenges, and a case study. *IEEE access*, 5:9882–9910, 2017.

[30] Paola G Vinueza Naranjo, Enzo Baccarelli, and Michele Scarpiniti. Design and energy-efficient resource management of virtualized networked fog architectures for the real-time support of iot applications. *The journal of Supercomputing*, 74(6):2470–2507, 2018.

[31] Seyed Hossein Mortazavi, Mohammad Salehe, Carolina Simoes Gomes, Caleb Phillips, and Eyal de Lara. Cloudpath: A multi-tier cloud computing framework. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, pages 1–13, 2017.

[32] Seyed Hossein Mortazavi, Bharath Balasubramanian, Eyal de Lara, and Shankaranarayanan Puzhavakath Narayanan. Pathstore, a data storage layer for the edge. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 519–519, 2018.

[33] Seyed Hossein Mortazavi, Mohammad Salehe, Bharath Balasubramanian, Eyal de Lara, and ShankaranarayananPuzhavakathNarayanan. Sessionstore: A session-aware datastore for the edge. In *2020 IEEE 4th International Conference on Fog and Edge Computing (ICFEC)*, pages 59–68. IEEE, 2020.

[34] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[35] Harshit Gupta, Zhuangdi Xu, and Umakishore Ramachandran. Datafog: Towards a holistic data management platform for the iot age at the network edge. In {*USENIX*} *Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[36] Ruben Mayer, Harshit Gupta, Enrique Saurez, and Umakishore Ramachandran. Fogstore: Toward a distributed data store for fog computing. In *2017 IEEE Fog World Congress (FWC)*, pages 1–6. IEEE, 2017.

[37] David Reinsel, John Gantz, and John Rydning. Data age 2025: The evolution of data to life-critical. *Don't Focus on Big Data*, 2, 2017.

[38] Qualcomm Technologies Inc. Making immersive virtual reality possible in mobile. `https://www.qualcomm.com/media/documents/files/whitepaper-making-immersive-virtual-reality-possible-in-mobile.pdf`, 2016. Accessed on 2022-09-24.

[39] Piero A Bonatti and Sabrina Kirrane. Big data and analytics in the age of the gdpr. In *2019 IEEE International Congress on Big Data (BigDataCongress)*, pages 7–16. IEEE, 2019.

[40] Mohammed Islam Naas, Philippe Raipin Parvedy, Jalil Boukhobza, and Laurent Lemarchand. ifogstor: an iot data placement strategy for fog infrastructure. In *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pages 97–104. IEEE, 2017.

[41] Tiansheng Huang, Weiwei Lin, Yin Li, LiGang He, and ShaoLiang Peng. A latency-aware multiple data replicas placement strategy for fog computing. *Journal of Signal Processing Systems*, 91(10):1191–1204, 2019.

[42] Nick Roussopoulos, Stephen Kelley, and Frederic Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 71–79, 1995.

[43] Claudio Marche, Luigi Atzori, Virginia Pilloni, and Michele Nitti. How to exploit the social internet of things: Query generation model and device profiles' dataset. *Computer Networks*, page 107248, 2020.

[44] Dirk G Cattrysse and Luk N Van Wassenhove. A survey of algorithms for the generalized assignment problem. *European journal of operational research*, 60(3):260–272, 1992.

[45] Mohammed Islam Naas, Laurent Lemarchand, Jalil Boukhobza, and Philippe Raipin. A graph partitioning-based heuristic for runtime iot data placement strategies in a fog infrastructure. In *Proceedings of the 33rd annual ACM symposium on applied computing*, pages 767–774, 2018.

[46] Mohammed Islam Naas, Laurent Lemarchand, Philippe Raipin, and Jalil Boukhobza. Iot data replication and consistency management in fog computing. *Journal of Grid Computing*, 19(3):1–25, 2021.

[47] Nenad Mladenović, Jack Brimberg, Pierre Hansen, and José A Moreno-Pérez. The p-median problem: A survey of metaheuristic approaches. *European Journal of Operational Research*, 179(3):927–939, 2007.

[48] Bing Lin, Fangning Zhu, Jianshan Zhang, Jiaqing Chen, Xing Chen, Naixue N Xiong, and Jaime Lloret Mauri. A time-driven data placement strategy for a scientific workflow combining edge computing and cloud computing. *IEEE Transactions on Industrial Informatics*, 15(7):4254–4265, 2019.

[49] Xin Du, Songtao Tang, Zhihui Lu, Jie Wet, Keke Gai, and Patrick CK Hung. A novel data placement strategy for data-sharing scientific workflows in heterogeneous edge-cloud computing environments. In *2020 IEEE International Conference on Web Services (ICWS)*, pages 498–507. IEEE, 2020.

[50] Junjie Xie, Chen Qian, Deke Guo, Minmei Wang, Shouqian Shi, and Honghui Chen. Efficient indexing mechanism for unstructured data sharing systems in edge computing. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 820–828. IEEE, 2019.

[51] Xinliang Wei and Yu Wang. Popularity-based data placement with load balancing in edge computing. *IEEE Transactions on Cloud Computing*, 2021.

[52] Chunlin Li, Jingpan Bai, and JianHang Tang. Joint optimization of data placement and scheduling for improving user experience in edge computing. *Journal of Parallel and Distributed Computing*, 125:93–105, 2019.

[53] Karim Sonbol, Öznur Özkasap, Ibrahim Al-Oqily, and Moayad Aloqaily. Edgekv: decentralized, scalable, and consistent storage for the edge. *Journal of Parallel and Distributed Computing*, 144:28–40, 2020.

[54] Alistair C Veitch, Erik Riedel, Simon J Towers, John Wilkes, et al. Towards global storage management and data placement. In *HotOS*, page 184. Citeseer, 2001.

[55] Zheng Zhang, Mallik Mahalingam, Zhichen Xu, and Wenting Tang. Scalable, structured data placement over p2p storage utilities. In *Proceedings. 10th IEEE International Workshop on Future Trends of Distributed Computing Systems, 2004. FTDCS 2004.*, pages 244–251. IEEE, 2004.

[56] Tevfik Kosar and Miron Livny. Stork: Making data placement a first class citizen in the grid. In *24th International Conference on Distributed Computing Systems, 2004. Proceedings.*, pages 342–349. IEEE, 2004.

[57] Hsiangkai Wang, Pangfeng Liu, and Jan-Jan Wu. A qos-aware heuristic algorithm for replica placement. In *2006 7th IEEE/ACM International Conference on Grid Computing*, pages 96–103. IEEE, 2006.

[58] André Brinkmann, Sascha Effert, Friedhelm Meyer auf der Heide, and Christian Scheideler. Dynamic and redundant data placement. In *27th International Conference on Distributed Computing Systems (ICDCS'07)*, pages 29–29. IEEE, 2007.

[59] Bassam A Alqaralleh, Chen Wang, Bing Bing Zhou, and Albert Y Zomaya. Effects of replica placement algorithms on performance of structured overlay networks. In *2007 IEEE International Parallel and Distributed Processing Symposium*, pages 1–8. IEEE, 2007.

[60] Yongqiang He, Rubao Lee, Yin Huai, Zheng Shao, Namit Jain, Xiaodong Zhang, and Zhiwei Xu. Rcfile: A fast and space-efficient data placement structure in mapreduce-based warehouse systems. In *2011 IEEE 27th International Conference on Data Engineering*, pages 1199–1208. IEEE, 2011.

[61] Sharrukh Zaman and Daniel Grosu. A distributed algorithm for the replica placement problem. *IEEE Transactions on Parallel and Distributed Systems*, 22(9):1455–1468, 2011.

[62] Hassan I Abdalla. An efficient approach for data placement in distributed systems. In *2011 Fifth FTRA international conference on multimedia and ubiquitous engineering*, pages 297–301. IEEE, 2011.

[63] Wei Dai, Ibrahim Ibrahim, and Mostafa Bassiouni. A new replica placement policy for hadoop distributed file system. In *2016 IEEE 2nd international conference on big data security on cloud (bigdatasecurity), IEEE international conference on high performance and smart computing (HPSC), and IEEE international conference on intelligent data and security (IDS)*, pages 262–267. IEEE, 2016.

[64] Swaminathan Sivasubramanian. Amazon dynamodb: a seamlessly scalable non-relational database service. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 729–730, 2012.

[65] Voldemort project. `https://www.project-voldemort.com/voldemort/`, 2013. Accessed on 2022-09-25.

[66] Rahul Islam, Vasco Xu, and Karan Ahuja. Motiontrace: Imu-based field of view prediction for smartphone ar interactions. *arXiv preprint arXiv:2408.01850*, 2024.

[67] Meraj Khan and Arnab Nandi. Dreamstore: A data platform for enabling shared augmented reality. In *2021 IEEE Virtual Reality and 3D User Interfaces (VR)*, pages 555–563. IEEE, 2021.

[68] Moatasim Mahmoud, Stamatia Rizou, Andreas S Panayides, Nikolaos V Kantartzis, George K Karagiannidis, Pavlos I Lazaridis, and Zaharias D Zaharis. A survey on optimizing mobile delivery of 360 videos: Edge caching and multicasting. *IEEE access*, 11:68925–68942, 2023.

[69] Leandro Ordonez-Ante, Jeroen van der Hooft, Tim Wauters, Gregory Van Seghbroeck, Bruno Volckaert, and Filip De Turck. Explora-vr: Content prefetching for tile-based immersive video streaming applications. *Journal of Network and Systems Management*, 30(3):38, 2022.

[70] Shuquan Liu, Guanghui Zhang, Mengbai Xiao, Dongxiao Yu, and Xiuzhen Cheng. An intelligent prefetch strategy with multi-round cell enhancement in volumetric video streaming. In *2024 21st Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 1–9. IEEE, 2024.

[71] Nan Wu, Kaiyan Liu, Ruizhi Cheng, Bo Han, and Puqi Zhou. Theia: Gaze-driven and perception-aware volumetric content delivery for mixed reality headsets. In *Proceedings of the 22nd Annual International Conference on Mobile Systems, Applications and Services*, pages 70–84, 2024.

[72] Seyun Choi, Sukjun Hong, Hoijun Kim, Seunghyun Lee, and Soonchul Kwon. Prefetching method for low-latency web ar in the wmn edge server. *Applied Sciences*, 13(1):133, 2022.

[73] Pixcap. Glb file: What is it, how to create and open it? Accessed: 2025-04-27.

[74] Erna Hikmawati, Nur Ulfa Maulidevi, and Kridanto Surendro. Minimum threshold determination method based on dataset characteristics in association rule mining. *Journal of Big Data*, 8:1–17, 2021.

[75] Schutz, Felix and Roman, Adrian . Simcpp20. Accessed: 2025-04-27.

[76] Sebastian Raschka. Mlxtend: Providing machine learning and data science utilities and extensions to python's scientific computing stack. *The Journal of Open Source Software*, 3(24), April 2018.

[77] Philippe Fournier-Viger. Spmf datasets. Accessed: 2025-04-27.

[78] Udacity. Fcnd-motion-planning. Accessed: 2025-04-27.

[79] Viet Nguyen. The state of 5g: Growth, challenges, and opportunities in 2025. Accessed: 2025-04-27.

[80] AWS. Aws direct connect. Accessed: 2025-04-27.

[81] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13142–13153, 2023.

[82] Yushan Siriwardhana et al. A survey on mobile augmented reality with 5g mobile edge computing: Architectures, applications, and technical aspects. *IEEE Commun. Surv. Tutor.*, 23:1160–1192, 2021.

[83] Jinke Ren et al. An edge-computing based architecture for mobile augmented reality. *IEEE Network*, 33(4):162–169, 2019.

[84] Nuno Pereira et al. Arena: The augmented reality edge networking architecture. In *ISMAR*, pages 479–488. IEEE, 2021.

[85] Olivier Debauche, Saïd Mahmoudi, and Adriano Guttadauria. A new edge computing architecture for iot and multimedia data management. *Information*, 13(2):89, 2022.

[86] Chien-Cheng Wu et al. On local cache management strategies for mobile augmented reality. In *WoWMoM*, pages 1–3. IEEE, 2014.

[87] Wenxiao Zhang et al. Cloudar: A cloud-based framework for mobile augmented reality. In *Thematic Workshops of ACM Multimedia 2017*, pages 194–200, 2017.

[88] Zhanpeng Huang et al. Cloudridar: A cloud-based architecture for mobile augmented reality. In *MARS*, pages 29–34, 2014.

[89] Xiuquan Qiao et al. Web ar: A promising future for mobile augmented reality—state of the art, challenges, and insights. *Proceedings of the IEEE*, 107(4):651–666, 2019.

[90] Theodoros Theodoropoulos et al. Cloud-based xr services: A survey on relevant challenges and enabling technologies. *J. Netw. Netw. Appl.*, 2(1):1–22, 2022.

[91] Yuhan Zhao et al. A survey on caching in mobile edge computing. *Wirel. Commun. Mob. Comput.*, 2021(1):5565648, 2021.

[92] Sukhmani Sukhmani et al. Edge caching and computing in 5g for mobile ar/vr and tactile internet. *IEEE MultiMedia*, 26(1):21–30, 2018.

[93] Utsav Drolia et al. Cachier: Edge-caching for recognition applications. In *ICDCS*, pages 276–286. IEEE, 2017.

[94] Raluca Halalai et al. Agar: A caching system for erasure-coded data. In *ICDCS*, pages 23–33. IEEE, 2017.

[95] Xiaoyu Xia et al. Online collaborative data caching in edge computing. *IEEE Trans. Parallel Distrib. Syst.*, 32(2):281–294, 2020.

[96] Wenxiao Zhang et al. Cars: Collaborative augmented reality for socialization. In *ACM HotMobile*, pages 25–30, 2018.

[97] Wenxiao Zhang, Bo Han, and Pan Hui. Sear: Scaling experiences in multi-user augmented reality. *IEEE Trans. Vis. Comput. Graph.*, 28(5):1982–1992, 2022.

[98] Anahita Mahzari et al. Coopec: Cooperative prefetching and edge caching for adaptive 360° video streaming. In *ISM*, pages 77–81. IEEE, 2020.

[99] Pantelis Maniotis and Nikolaos Thomos. Tile-based edge caching for 360° live video streaming. *IEEE Trans. Circuits Syst. Video Technol.*, 31(12):4938–4950, 2021.

[100] Dongbiao He et al. Cubist: High-quality 360-degree video streaming services via tile-based edge caching and fov-adaptive prefetching. In *ICWS*, pages 208–218. IEEE, 2021.

[101] Jesús Aguilar-Armijo et al. Space: Segment prefetching and caching at the edge for adaptive video streaming. *IEEE Access*, 11:21783–21798, 2023.

[102] Wanxin Shi et al. Leap: learning-based smart edge with caching and prefetching for adaptive video streaming. In *IWQoS*, pages 1–10, 2019.

[103] Feixiong Zhang et al. Edgebuffer: Caching and prefetching content at the edge in the mobilityfirst future internet architecture. In *WoWMoM*, pages 1–9. IEEE, 2015.

[104] Rakesh Agrawal et al. Mining association rules between sets of items in large databases. In *ACM SIGMOD*, pages 207–216, 1993.

[105] Rakesh Agrawal et al. Fast discovery of association rules. *Advances in knowledge discovery and data mining*, 12(1):307–328, 1996.

[106] Juncheng Yang et al. Mithril: mining sporadic associations for cache prefetching. In *ACM SoCC*, pages 66–79, 2017.

[107] Waleed Ali et al. A survey of web caching and prefetching. *Int. J. Advance. Soft Comput. Appl*, 3(1):18–44, 2011.

[108] João Passos, Sérgio Ivan Lopes, Filipe Manuel Clemente, Pedro Miguel Moreira, Markel Rico-González, Pedro Bezerra, and Luís Paulo Rodrigues. Wearables and internet of things (iot) technologies for fitness assessment: a systematic review. *Sensors*, 21(16):5418, 2021.

[109] Christopher C Wilmers, Barry Nickel, Caleb M Bryce, Justine A Smith, Rachel E Wheat, and Veronica Yovovich. The golden age of bio-logging: How animal-borne sensors are advancing the frontiers of ecology. *Ecology*, 96(7):1741–1753, 2015.

[110] Rob O'Dwyer. Shell trials crew smartwatches for safety management, 2022.

[111] Andrew G Kubala, Peter G Roma, Jason T Jameson, Pinata H Sessoms, Evan D Chinoy, Luis R Rosado, Trevor B Viboch, Brandon J Schrom, Hedaya N Rizeq, Prayag S Gordy, et al. Advancing a us navy shipboard infrastructure for sleep monitoring with wearable technology. *Applied Ergonomics*, 117:104225, 2024.

[112] Ekaterina Svertoka, Salwa Saafi, Alexandru Rusu-Casandra, Radim Burget, Ion Marghescu, Jiri Hosek, and Aleksandr Ometov. Wearables for industrial work safety: A survey. *Sensors*, 21(11):3844, 2021.

[113] Ramesh Kumar Sah and Hassan Ghasemzadeh. Minimum-cost sensor channel selection for wearable computing. *arXiv preprint arXiv:2402.00875*, 2024.

[114] NK Anish, Ganapati Bhat, Jaehyun Park, Hyung Gyu Lee, and Umit Y Ogras. Sensor-classifier co-optimization for wearable human activity recognition applications. In *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*, pages 1–4. IEEE, 2019.

[115] Elisa Borella, Umut Berk Çakmakçi, Enrico Gottardis, Alessandro Buratto, Thomas Marchioro, and Leonardo Badia. Effective sensor selection for human activity recognition via shapley value. In *2024 IEEE International Workshop on Metrology for Living Environment (MetroLivEnv)*, pages 22–27. IEEE, 2024.

[116] Yiming Tian and Jie Zhang. Optimizing sensor deployment for multi-sensor-based har system with improved glowworm swarm optimization algorithm. *Sensors*, 20(24):7161, 2020.

[117] Randy Ardywibowo, Shahin Boluki, Zhangyang Wang, Bobak J Mortazavi, Shuai Huang, and Xiaoning Qian. Vfds: variational foresight dynamic selection in bayesian neural networks for efficient human activity recognition. In *International Conference on Artificial Intelligence and Statistics*, pages 1359–1379. PMLR, 2022.

[118] Mohammad Malekzadeh, Richard Clegg, Andrea Cavallaro, and Hamed Haddadi. Dana: Dimension-adaptive neural architecture for multivariate sensor data. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 5(3):1–27, 2021.

[119] Alex L Wood, Geoff V Merrett, Steve R Gunn, Bashir M Al-Hashimi, Nigel R Shadbolt, and Wendy Hall. Adaptive sampling in context-aware systems: a machine learning approach. In *IET Conference on Wireless Sensor Systems (WSS 2012)*, pages 1–5. IET, 2012.

[120] Guangyu Yang, Lei Zhang, Can Bu, Shuaishuai Wang, Hao Wu, and Aiguo Song. Freqsense: Adaptive sampling rates for sensor-based human activity recognition under tunable computational budgets. *IEEE Journal of Biomedical and Health Informatics*, 27(12):5791–5802, 2023.

[121] Yassine Ben-Aboud, Daniel Bonilla Licea, Mounir Ghogho, and Abdellatif Kobbane. On adaptive sampling algorithms for iot devices. In *ICC 2021-IEEE International Conference on Communications*, pages 1–7. IEEE, 2021.

[122] Marco Giordano, Silvano Cortesi, Prodromos-Vasileios Mekikis, Michele Crabolu, Giovanni Bellusci, and Michele Magno. Energy-aware adaptive sampling for self-sustainability in resource-constrained iot devices. In *Proceedings of the 11th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems*, pages 65–71, 2023.

[123] Shih-Yeh Chen, Chin-Feng Lai, Ren-Hung Hwang, Ying-Hsun Lai, and Ming-Shi Wang. An adaptive sensor data segments selection method for wearable health care services. *Journal of medical systems*, 39(12):194, 2015.

[124] Xiaodong Yang, Yiqiang Chen, Hanchao Yu, Yingwei Zhang, Wang Lu, and Ruizhe Sun. Instance-wise dynamic sensor selection for human activity recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1104–1111, 2020.

[125] Mengxi Liu, Zimin Zhao, Daniel Geißler, Bo Zhou, Sungho Suh, and Paul Lukowicz. Coss: Co-optimizing sensor and sampling rate for data-efficient ai in human activity recognition. In *2nd Workshop on Sustainable AI*.

[126] Oscar D Lara and Miguel A Labrador. A survey on human activity recognition using wearable sensors. *IEEE communications surveys & tutorials*, 15(3):1192–1209, 2012.

[127] Lorenzo Scalise and Gloria Cosoli. Wearables for health and fitness: Measurement characteristics and accuracy. In *2018 IEEE international instrumentation and measurement technology conference (I2MTC)*, pages 1–6. IEEE, 2018.

[128] Graeme Mattison, Oliver Canfell, Doug Forrester, Chelsea Dobbins, Daniel Smith, Juha Töyräs, and Clair Sullivan. The influence of wearables on health care outcomes in chronic disease: systematic review. *Journal of medical Internet research*, 24(7):e36690, 2022.

[129] Sensing solutions for wearables, 2024. Accessed on 2026-02-05.

[130] Gary Weiss. WISDM Smartphone and Smartwatch Activity and Biometrics Dataset . UCI Machine Learning Repository, 2019. DOI: https://doi.org/10.24432/C5HK59.

[131] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[132] Oresti Banos, Rafael Garcia, Juan A Holgado-Terriza, Miguel Damas, Hector Pomares, Ignacio Rojas, Alejandro Saez, and Claudia Villalonga. mhealthdroid: a novel framework for agile development of mobile health applications. In *International workshop on ambient assisted living*, pages 91–98. Springer, 2014.

[133] Oresti Banos, C Villalonga, R Garcia, A Saez, M Damas, JA Holgado-Terriza, S Lee, H Pomares, and I Rojas. Design, implementation and validation of a novel open framework for agile development of mobile health applications. *BioMedical Engineering OnLine*, 14:1–20, 2015.

[134] Attila Reiss and Didier Stricker. Introducing a new benchmarked dataset for activity monitoring. In *2012 16th international symposium on wearable computers*, pages 108–109. IEEE, 2012.

[135] Sizhen Bian, Mengxi Liu, Vitor Fortes Rey, Daniel Geissler, and Paul Lukowicz. Tinierhar: Towards ultra-lightweight deep learning models for efficient human activity recognition on edge devices. *arXiv preprint arXiv:2507.07949*, 2025.

[136] Xin Qi, Matthew Keally, et al. Adasense: Adapting sampling rates for activity recognition in body sensor networks. In *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 163–172. IEEE, 2013.

[137] Bosch Sensortec. BMI160 Data sheet. Technical report, November 2020. Document revision 1.0, Document number BST-BMI160-DS000-09, Accessed on 2026-02-05.

[138] STMicroelectronics. LSM6DSO: 6-axis inertial sensor with high data rate. Technical report, June 2024. Document revision 3, Accessed on 2026-02-05.

[139] Bosch Sensortec. BMM350: 3-axis magnetic sensor with high data rate. Technical report, February 2025. Document revision 1.27, Document number BST-BMM350-DS001-27, Accessed on 2026-02-05.

[140] Analog Devices. MAX30003 - Ultra-Low Power, Single-Channel Integrated Biopotential (ECG, R-to-R Detection) AFE. Technical report, January 2018. Rev 0; 1/18, Accessed on 2026-02-05.

[141] NOVOSENSE Microelectronics. NST112 High Accuracy, Low-Power Temperature Sensor with SMBus and I2C Interface. Technical report, 2023. Rev. 2.1, Accessed on 2026-02-05.

[142] Nordic Semiconductor. nRF52840 Product Specification, 2018. Accessed on 2026-02-05.

[143] Xing Liu and Feng Qian. Measuring and optimizing android smartwatch energy consumption: poster. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, pages 421–423, 2016.

[144] Assim Boukhayma, Anthony Barison, Serj Haddad, and Antonino Caizzone. Ring-embedded micro-power mm-sized optical sensor for accurate heart beat monitoring. *IEEE Access*, 9:127217–127225, 2021.

[145] Hamed Rezaie and Mona Ghassemian. An adaptive algorithm to improve energy efficiency in wearable activity recognition systems. *IEEE Sensors Journal*, 17(16):5315–5323, 2017.

Appendix A

NP-Hardness Proof

Theorem 1. *The APSSE problem formulated in equations* (5.4)–(5.7) *is NP-Hard.*

Proof. We prove this by reduction from the **Weighted Set Cover (WSC)** problem, which is known to be NP-Hard.

The Weighted Set Cover Problem: Given a universe of elements $U = \{u_1, u_2, \ldots, u_m\}$ (elements to be covered) and a collection of subsets $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ where each $S_i \subseteq U$ has an associated cost w_i. The goal is to select a sub-collection of sets such that their union covers all elements in U and the total cost is minimized.

Mapping to APSSE: We construct a special instance of the APSSE problem that is isomorphic to the WSC instance:

1. Mapping Activities to Universe: Let the Universe U represent the set of activity types required. The condition $C(x) \leq C_{threshold}$ is equivalent to covering all necessary activity types (i.e., achieving 100% required accuracy implies covering all elements in U).

2. Mapping Sensors to Sets: Let each sensor s represent a set S_i in the WSC problem.

3. Mapping Sampling Rates to Selection: Restrict each sensor s to have only two rates: $R_s = \{r_{s,0}, r_{s,1}\}$ where $r_{s,0} = 0$ and $r_{s,1} = r_{high}$ for simplicity.

    - Choosing index $j = 0$ (OFF) corresponds to *not selecting* set S_i. Cost = 0.
    - Choosing index $j = 1$ (ON) corresponds to *selecting* set S_i. Cost = $E(s, r_{s,1}) = w_i$.

4. Mapping Accuracy to Coverage: We define the error function $C(x)$ such that $C(x) \leq C_{threshold}$ is satisfied *if and only if* the union of features provided by the selected active sensors fully covers the universe U. If any element $u \in U$ is uncovered, the error remains above the threshold.

Conclusion: Finding the minimum energy configuration for APSSE in this instance is equivalent to finding the minimum weight collection of sets that covers U. Since Weighted Set Cover is NP-Hard and it can be reduced to a specific instance of APSSE, the general APSSE problem is also NP-Hard. □

Remark 1. *The formulated APSSE problem addresses the selection of a static configuration x. The problem of per-activity adaptive selection (finding a mapping from activities to configurations) is a generalization of this static case. Since the static problem (where the mapping is constrained to be constant) is NP-Hard, the general per-activity optimization problem is also NP-Hard.*