UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Reo Kuchida

Edge Intelligence on Command

Master's Thesis (30 ECTS)

Supervisors: Mayowa Olapade, MSc
Huber Flores, PhD

Tartu 2025

Edge Intelligence on Command

Abstract:
The deployment of distributed machine learning (DML) at the edge has introduced a new generation of intelligent, real-time applications in domains such as autonomous vehicles, pervasive robotics, smart environments, and ambient sensing. However, managing such infrastructures remains inherently complex, particularly in decentralized, resource-constrained, and dynamically changing environments populated by heterogeneous devices. Existing solutions often rely on static configurations or require expert knowledge for setup, orchestration, and maintenance, thereby limiting their accessibility and scalability in real-world contexts. This thesis introduces *Edge Intelligence on Command*, a novel architecture that employs large language models (LLMs) to initiate decentralized machine learning workflows via natural language intent. By combining prompt-engineered LLM agents with opportunistic device discovery and configuration, the system eliminates the need for specialized knowledge, empowering non-experts to deploy and manage ML tasks seamlessly. The architecture supports both federated and split learning, facilitating privacy-preserving, resource-efficient collaboration across diverse edge nodes. Beyond orchestration, the system addresses a critical challenge in DML deployments, which is ensuring the integrity of the learning process in the presence of abnormal device behavior. In particular, it focuses on poisoning attacks, where compromised or faulty edge nodes introduce corrupted data that can degrade model performance. In order to investigate this, a feasibility study conducted with a Raspberry Pi and thermal imaging reveals that poisoned training data induces detectable shifts in runtime behavior, including elevated temperature and increased CPU usage. These observations motivate the introduction of the Device Change Point Index (DCPI), a lightweight, decentralized anomaly detection mechanism based on native device metrics. Without relying on external hardware or centralized oversight, DCPI makes real-time trust assessment in edge learning environments. Taken together, this work demonstrates the feasibility and effectiveness of combining LLM-driven orchestration with runtime behavioral monitoring to enable secure, adaptive, and user-centric distributed intelligence at the edge. It contributes a practical step toward more autonomous and accessible edge AI systems.

Keywords:
Edge Intelligence (EI), Distributed Machine Learning (DML), Large Language Models (LLMs), Natural Language Interfaces, AI Orchestration, Anomaly Detection, Device Behavior Monitoring, Poisoning Attacks, AI systems, Decentralized Systems, Prompt Engineering

CERCS: P170 - Computer science, numerical analysis, system, control

Edge-intellekt käsu peale

Lühikokkuvõte:

Hajutatud masinõppe (DML, ingl k distributed machine learning) kasutamine edge-süsteemides on loonud soodsa pinnase uudsete nutikate ja reaalajas toimivate rakenduste arenguks, isejuhtivad sõidukid, ümbrus-teadlikud robootika, nutikad keskkonnad, ning andurpõhine mnitooring Edge-süsteemide haldamine on aga keerukas, kuna kasutatavad seadmed on sageli piiratud arvutusvõime ja energiavarudega ning süsteemi koosseisu kuuluvad seadeldised võivad olla väga erineva iseloomuga. Seetõttu tuginevad paljud tänapäevased süsteemid üksnes kitsale, ette määratud seadmevalikule või nõuavad nad ekspertteadmisi seadistamiseks, seadmetevahelise koostöö korraldamiseks ning süsteemide hoolduseks. Need piirangud vähendavad edge-süsteemide kättesaadavust ja takistavad nende praktilist, ulatuslikku rakendamist. Käesolev lõputöö tutvustab uudset arhitektuuri nimega Edge-intellekt käsu peale, mis rakendab suurkeelmudeleid (LLM, ingl k large language model), hajutatud masinõppe töövoogude algatamiseks tavakeelepõhiste käskluste alusel. Ühildades spetsiaalsete päringute abil loodud LLM agente, võimekusega oportunistlikult tuvastada ja sätestada seadeldisi, võimaldab antud arhitektuur ka ilma spetsialiseeritud teadmistega mitte-ekspertidel hõlpsasti luua ja hallata masinõppel põhinevaid ülesandeid. Arhitektuur toetab kahte tüüpi masinõpet: hajutatud õpet (ingl k federated learning) ja jaotatud õpet (ingl k split learning), võimaldades seadmetevaheline koostöö toimimist privaatsust-tagavalt ja energiatõhusalt. Lisaks seadmetevaheliste koostöö korraldamisele, käsitleb antud süsteem üht keskset probleemi DML-s: kuidas tagada masinõppes õppimisprotsessi usaldusväärsus ka juhtudel, kus mõni seadeldis käitub valesti või on sattunud pahatahtliku rünnaku alla. Peaasjalikult keskendutakse antud töös andmete mürgitamisele (ingl k poisoning attacks), kus rünnaku alla sattunud seadeldis saadab vigaseid andmeid, mille tagajärjel võib kogu süsteemi töö halveneda. Selle probleemi uurimiseks teostasime eksperimendi kasutades Raspberry Pi-de ja termokaamerate abil, mille käigus selgus, et mürgitatud andmed suurendavad mõõdetavalt nii seadme temperatuuri kui ka protsessori töökoormust. Selle põhjal rakendasime DCPI-d (ingl k Device Change Point Index), mis on vähenõudlik, detsentraliseeritud ja reaalajas toimiv kõrvalekallete tuvastamise mehhanism. Kuna DCPI kasutab seadme sisemisi mõõdikuid, siis ei ole selle kasutamiseks vaja lisaseadmeid ega keskset juhtimist. Kokkuvõttes, antud töö näitab LLM-põhise, reaalajas toimiva ja turvalise edge-süsteemide haldamise arhitektuuride teostatavust. See võimaldab luua turvalisi, paindlikke ja kasutajasõbralikke tehisintellektil põhinevaid rakendusi edge-süsteemides, tuues meid sammu võrra lähemale autonoomsetele ja kättesaadavatele edge-intellekt süsteemidele.

Võtmesõnad:

Edge-intellekt (EI), Hajutatud masinõpe (DML), suurkeelmudelid (LLM), tavakeelelised kasutajaliidesed, AI-põhine seadmete koostöö korraldus, kõrvalekallete tuvastamine, seadmete käitumise monitooring, andmete mürgitus, AI-põhised süsteemid, detsentraliseeritud süsteemid, päringuinseneeria

CERCS: P170 - Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Contents

1 Introduction

Distributed and decentralized systems coordinate multiple devices to collaborate toward shared goals. These systems can leverage device proximity for efficient sensing, computing, and networking tasks [14]. Recent advances have led to new paradigms of Edge Intelligence, which bring high-performance AI capabilities closer to users through local sensing, lightweight data preprocessing [41], and real-time inference with compact AI models [12]. It plays an essential role in supporting applications that require real-time and distributed decision-making, including autonomous vehicles, IoT sensor networks, and drone systems. However, the complexity of managing those systems, such as device heterogeneity and asynchronous participation, hinders the broader adoption. Achieving user-friendly collaborative inference across edge infrastructure remains a critical challenge, requiring new methods that simplify dynamic system formation while ensuring secure, flexible collaboration.

To address this challenge, existing applications have typically been developed as specialized clients. These clients are designed to manage the complexities of decentralized systems while supporting secure communication, robust data management, and efficient task execution. Examples include P2P cryptocurrencies such as Ethereum [87], federated learning applications such as Google's mobile keyword prediction [51], and distributed file sharing platforms such as BitTorrent [13]. Although these systems simplify operations for users, they often tie applications to fixed configurations and infrastructures, limiting flexibility and missing the opportunistic advantages of decentralized systems, such as leveraging nearby devices for dynamic collaboration. To improve usability, various solutions have been proposed to enhance user awareness of system formation and the direct benefits from collaboration, e.g., visual cues and notifications [55]. By increasing awareness, users can better understand the dynamic nature of the system, helping them make informed decisions that optimize resource utilization and performance. However, these solutions still often require manual intervention or technical expertise, creating barriers for widespread adoption. At this point, advances in LLM technologies can simplify user involvement by encapsulating technical complexities, leading to the dynamic formation of systems that seamlessly trigger interconnectivity and resource allocation [39]. This creates a significant opportunity to accelerate the development of decentralized applications and intelligence, harnessing their opportunistic nature while being guided by user commands using natural language.

In addition to this, ML training and inference capabilities across heterogeneous edge devices during collaborative tasks have been shown to be vulnerable to threats that can disrupt the optimal performance of such formation [4]. Recent studies have also shown that robust defense mechanisms in decentralized ML, particularly for AI inference tasks, can be used against vulnerability attacks, e.g., poisoning attacks [83]. This has been achieved through various approaches in Federated learning (FL), Split learning

(SL), and Split federated learning (SFL) [76]. For example, in FL, techniques such as robust aggregation and Byzantine-resilient algorithms help mitigate poisoning attacks by filtering out anomalous updates [6]. These methods significantly improve the security and reliability of distributed learning systems. However, many of these approaches require computationally intensive frameworks and deep technical intervention, which are rarely available in decentralized edge settings, impeding seamless integration and scalability.

In order to address the lack of flexible, dynamic coordination, this thesis proposes a unified framework that leverages LLMs to facilitate the dynamic formation and management of decentralized edge infrastructures. Using both proprietary and open source models of LLMs, we demonstrate how AI-based chatbots can assist users in assembling distributed computing setups by reasoning over device specifications, such as those collected from a popular dataset of smartphone models. These chatbots equipped with discovery can infer optimal configurations for executing opportunistic collaborative tasks (e.g, computing and sensing) and their outputs validated through human inspection. Additionally, we ensure that ML training and inference quality can be achieved while preserving the integrity of such collaborative tasks by introducing a lightweight, model-agnostic mechanism for detecting abnormal behavior of devices without requiring access to the central server or client-side models. With this mechanism, we monitor device-level execution metrics to detect anomalies caused by *poisoned data*, accounting for performance variability across heterogeneous hardware. This operates efficiently across different devices and learning paradigms. Together, our chatbot-driven orchestration and defense mechanism provide a scalable and secure foundation for the deployment of AI at the edge, with applications in drones, autonomous vehicles, and IoT systems. Our findings show both the promise and current limitations of LLMs in managing edge infrastructure within decentralized environments, while also revealing how device-level behavioral factors can significantly influence the reliability of edge intelligence systems.

1.1 Contributions

The following sums up the contributions:

- **Novel method:** We develop a unified framework that uses LLM-based chatbots to dynamically assemble decentralized edge infrastructures from opportunistic discovery and subsequently detect poisoning attacks without server or model access.
- **Novel insights:** We show that LLMs can reason over device specifications to form optimal computing setups, and that performance monitoring metrics captured using our proposed DCPI metric can reveal anomalies across heterogeneous edge devices.

- **Improved performance:** We conduct comprehensive experiments demonstrating that our approach is scalable and accurate, outperforming manual and existing techniques in infrastructure assembly and subsequently can preserve the integrity of the network within federated and split learning paradigms.

1.2 Outline

The remainder of this thesis is structured as follows:

- Chapter 2 provides a comprehensive review of related literature, focusing on decentralized systems, edge intelligence, LLM reasoning, and mechanisms for distributed ML.
- Chapter 3 presents the feasibility and conceptual validity of using LLMs for optimal infrastructure formation.
- Chapter 4 details the methodology for both the LLM-based infrastructure formation and the ML-based performance monitoring framework.
- Chapter 5 presents the experimental setup and evaluation of the performance monitoring components.
- Chapter 6 discusses the key findings, comparative analysis, and insights derived from the experimental results.
- Chapter 7 explores the practical implications, limitations, and directions for future work.
- Chapter 8 concludes the thesis by summarizing the main contributions and findings.

Note: Part of this thesis has been accepted for publication in IEEE INFOCOM Workshops, the 2nd International Workshop on Integrating Edge Intelligence and Large Models in Next Generation Networks (IEILM), London 2025.

2 State of the Art

This Chapter provides an extensive review of recent technological advancements in DML and Edge Intelligence architectures, focusing on strategies for managing edge infrastructures in a scalable manner. This begins by reviewing the evolution of decentralized systems and DML systems in facilitating real-time and communication-efficient collaboration. This provides insights into the analysis of various technologies, such as AI-Chatbots, autonomous agents, and rule-based frameworks that collaboratively contribute to the orchestration of heterogeneous edge resources. Another key component of this review is the assessment of previous solutions that have aimed to establish reliable ML training pipelines. In particular, it examines approaches designed to mitigate vulnerabilities such as poisoning attacks, ensuring the integrity of distributed learning processes in dynamic and potentially adversarial environments. By integrating insights from both the domains of user-centric infrastructure orchestration and secure ML training, this chapter seeks to identify common shortcomings in existing methodologies and to propose comprehensive solutions that guarantee both scalability and robustness. This dual focus of assembling user-friendly edge infrastructures and mitigating ML training against malicious interference during collaborative tasks will form the basis of this thesis, guiding subsequent investigations into novel techniques designed to enhance the overall performance of decentralized systems.

2.1 Decentralized Systems And Edge Intelligence

2.1.1 Overview and Evolution

Decentralized systems represent a paradigm shift from traditional client-server architectures to distributed networks where computational resources, data storage, and decision-making capabilities are distributed across geographically distributed nodes [75]. This evolution began with early peer-to-peer (P2P) networks such as Napster and matured with blockchain technology [54], which introduced trustless consensus mechanisms. These systems consist of autonomous entities (peers) that operate independently and interact with each other to achieve their individual goals, without coordination from a central authority. This autonomy provides flexibility, as peers can join or leave freely. These systems are known to improve scalability and avoid a single point of failure by distributing responsibility between multiple nodes [73]. In addition to their decentralized architecture, these systems also support context-aware and opportunistic coordination [14], allowing them to perform well in changing conditions and limited infrastructure. For example, nearby devices can form temporary clusters to handle intensive tasks locally, reducing latency. Similarly, multi-device applications can share GPS data, storage, or bandwidth to improve performance and energy efficiency [21].

Edge computing, sometimes called **fog computing**, refers to a decentralized infras-

tructure that extends cloud services to the periphery of the network, placing compute, storage, and networking closer to the end devices [9]. Processing data on the edge reduces latency, reduces bandwidth usage, and improves energy efficiency. Typical infrastructures include IoT gateways, mobile base stations, and micro-data centers that support local processing and minimize cloud dependency. This shift enables edge devices to take on tasks previously handled in the cloud, enhancing scalability and responsiveness [70].

Building on this foundation, **Edge Intelligence** has emerged as a paradigm that brings AI capabilities closer to end users. It combines local sensing, lightweight data preprocessing, and real-time inference using compact models, driven by the proliferation of IoT devices (expected to exceed 29 billion by 2030 [72]) and the demand for latency-sensitive applications [91]. Edge Intelligence frameworks often incorporate fog computing [99] and cloudlets [65] to allow more responsive and localized decision-making. Technologies such as TinyML [84] support on-device learning in constrained environments, enabling deployment in domains such as smart cities, vehicular networks, and industrial IoT [91]. Figure 1 illustrates several architectural examples, including (a) server-assisted learning, (b) offloading to nearby powerful devices, and (c) peer-to-peer collaboration among heterogeneous edge nodes.

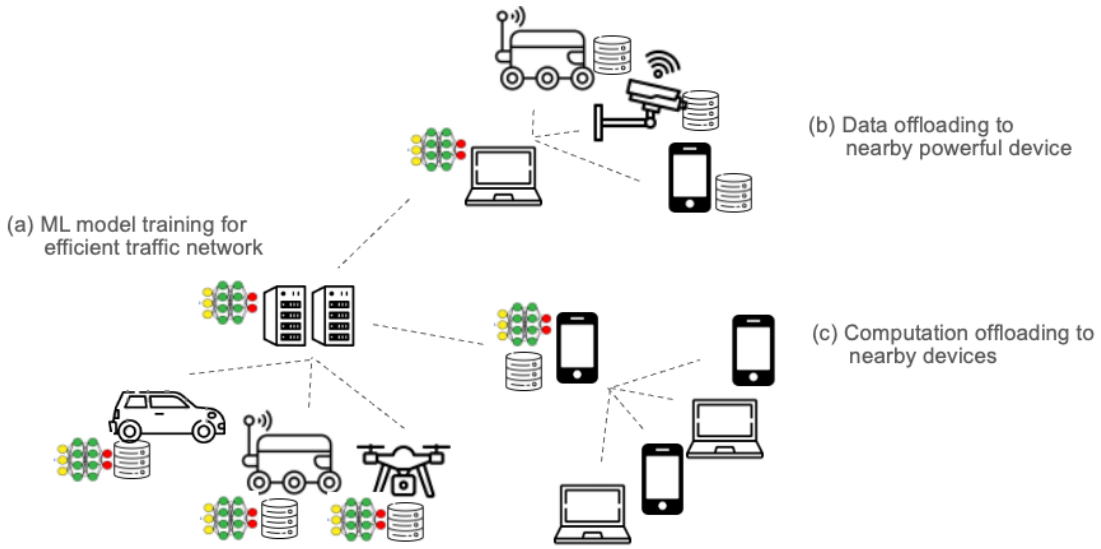


Figure 1. Illustration of Edge Intelligence in a heterogeneous environment, where various edge devices perform local computation with local data and collaborate with nearby nodes or edge servers.

2.1.2 Benefits and Motivation of Edge Intelligence

Edge Intelligence enhances decentralized systems by enabling AI processing directly at the data source, bringing several critical benefits for latency-sensitive, privacy-preserving, and bandwidth-limited applications. Processing data locally greatly reduces delay, a key requirement for real-time systems such as autonomous vehicles, where sensor fusion and decision making must occur in less than 100 milliseconds [47]. In addition, keeping data on the device strengthens privacy and security by avoiding the transmission of sensitive data (e.g., biometric data) over untrusted networks. This helps reduce the risk of attacks, such as data interception or unauthorized access, and supports compliance with regulatory frameworks such as the GDPR, which restricts personal data transfer across borders, and HIPAA, which governs medical data confidentiality in the U.S. [92]. Edge Intelligence also improves bandwidth efficiency by allowing edge nodes to compress, filter, or summarize sensor data before uploading to the cloud. This can reduce transmission and cloud storage costs significantly, by up to 70% in large-scale IoT deployments. Furthermore, decentralized architectures offer inherent fault tolerance. Even if cloud connectivity is lost, local clusters can continue to operate using microservices or federated learning setups. Mesh networking and distributed consensus protocols such as Raft and Paxos allow systems to reroute tasks dynamically, maintaining service availability and meeting service level agreement (SLA) requirements even when parts of the network fail.

2.1.3 Challenges and Limitations of Edge Intelligence

Despite its advantages, the real-world deployment of Edge Intelligence remains challenging due to a combination of technical, architectural, and environmental factors. These include not only heterogeneity [91] in hardware and software but also intermittent connectivity, security vulnerabilities, rigid deployment models, and the unpredictability of ad hoc networking scenarios. In the following, we outline several key challenges faced in deploying Edge Intelligence in practice.

Heterogeneity among edge devices: Edge environments include resource-constrained nodes with diverse compute profiles (e.g., Raspberry Pis vs. FPGA-accelerated gateways), OS kernels (e.g., real-time RTOS vs. Linux), and network protocols. Creating and deploying standardized ML pipelines is complex due to varying CPU/GPU performance, thermal throttling, memory constraints, and inconsistent I/O speeds resulting from different sensor sampling rates or storage latencies [98].

Unreliable or intermittent connectivity: Network instability makes coordination between devices more difficult. Many distributed ML frameworks assume reliable and synchronized communication between all participants. In reality, devices may drop in and out of connection, leading to inconsistencies in the model or outdated updates. Robust protocols such as Byzantine fault tolerance [10] may be required to mitigate

these failures or detect malicious participants, but often introduce high overhead due to complex consensus and cryptographic mechanisms.

Security vulnerabilities in edge training: Edge devices are frequently deployed in open or unprotected environments, making them targets for attacks. Adversaries can manipulate input data during training (poisoning), inject fake sensor readings to mislead inference (adversarial spoofing), or extract sensitive features through side-channel attacks like power analysis. These threats are particularly relevant during local model training and aggregation in edge settings, where centralized monitoring is often absent.

Rigid client-server topologies: Many existing edge applications rely on fixed topologies and purpose-built client software tightly coupled to specific platforms, device types, or protocols. Although these are straightforward to implement, they lack portability and adaptability, making it difficult to extend collaboration across diverse environments [2].

Opportunistic ad hoc networks: In highly dynamic scenarios, such as drone swarms or vehicular networks, devices frequently enter and leave the system. These environments require lightweight discovery and routing protocols that can be adapted in real time. However, most current platforms prioritize vertical integration (e.g., Azure IoT Edge) over horizontal flexibility, limiting the scalability of decentralized collaboration [16].

These challenges underscore the need for more flexible, intelligent, and adaptive orchestration frameworks that can operate reliably in heterogeneous, dynamic edge environments.

2.1.4 Recent Advances and Trends

Recent advances in Edge Intelligence reflect a shift toward architectures that are more autonomous, resilient, and capable of operating under real-time, privacy-sensitive, and resource-constrained conditions. A major direction is the use of blockchain and distributed ledger technologies to establish trust and ensure immutable audit trails across decentralized edge networks [1]. This is particularly important for multi-stakeholder systems that require secure interaction without centralized control. In parallel, collaborative edge learning frameworks, such as peer-to-peer and gossip-based schemes, have emerged to reduce dependence on centralized aggregation servers [96]. These approaches improve scalability and robustness by distributing learning responsibilities more evenly across nodes. Agent-based control and software-defined networking (SDN) are also being integrated into edge platforms to enable dynamic resource provisioning and fine-grained policy enforcement [67]. These technologies allow systems to adapt in real time to contextual changes, which is critical for latency-sensitive and mission-critical applications. Finally, hardware-oriented trends such as TinyML [84] and neuromorphic accelerators [49] are pushing AI inference capabilities directly onto low-power microcontrollers. This expands the reach of Edge Intelligence to highly constrained devices and broadens the deployment landscape for real-world edge AI systems.

2.2 Distributed Machine Learning Systems

2.2.1 Core Principles and Approaches

DML refers to the training of ML models across multiple devices or nodes without centralizing the data or computing. This approach is particularly useful when data are too large to transfer, geographically distributed, or privacy-sensitive [82]. DML enables parallel computation and improves scalability while addressing constraints in bandwidth and data governance. A well-known DML paradigm is FL, which allows devices to train models locally using their own data and share only model updates with a central server for aggregation [50]. FL enhances data privacy and reduces communication overhead, making it effective for applications such as smartphone personalization or healthcare. However, FL requires each device to train the entire model, which may be infeasible for resource-limited devices such as IoT sensors or low-end smartphones. In contrast, SL partitions the model into two segments: the client processes the initial layers, while the server handles the remaining ones [81]. This design reduces the computational burden on clients and also improves model privacy, as clients never access the full model. However, a key limitation is that SL typically uses a sequential relay-based training process, where only one client interacts with the server at a time [76]. SFL combines the parallelism of FL with the architectural separation of SL. In SFL, each client runs the front part of the model independently, while a shared server handles the back-end layers and aggregates updates from multiple clients in parallel. This hybrid design balances computational efficiency and privacy, making it well-suited for collaborative edge learning scenarios. Figure 2 compares these approaches, highlighting the distribution of computation and communication across the FL, SL, and SFL architectures.

As DML systems are increasingly deployed in edge environments, the design of training architectures must account for device-level constraints and heterogeneity [91]. Unlike traditional distributed training in data centers, edge training occurs on devices with limited computational power, intermittent connectivity, and various hardware profiles. When a device is sufficiently capable, it can perform solo training by running the entire model locally. Early studies demonstrated the feasibility of this approach on smartphones for tasks such as activity recognition and audio detection [40]. However, in resource-limited settings, collaborative training is often necessary. A common architecture is the master-worker model, where an edge server (the master) coordinates with multiple client devices (workers). For example, in the DeepCham framework [42], mobile devices generate training instances, while the edge server trains an adaptation model. Alternatively, P2P training allows devices to interact symmetrically. Each device trains a partial model on its local data and then exchanges updates directly with other nodes. Studies have shown that this structure can achieve accuracy comparable to centralized training in applications such as pattern and activity recognition [77]. To further enhance training efficiency under edge constraints, specialized methods have been proposed.

RecycleML [90], for example, applies cross-modal knowledge transfer using a shared architecture across sensing modalities. This allows devices to reuse features learned from one modality to accelerate training in another, especially useful when labeled data is sparse.

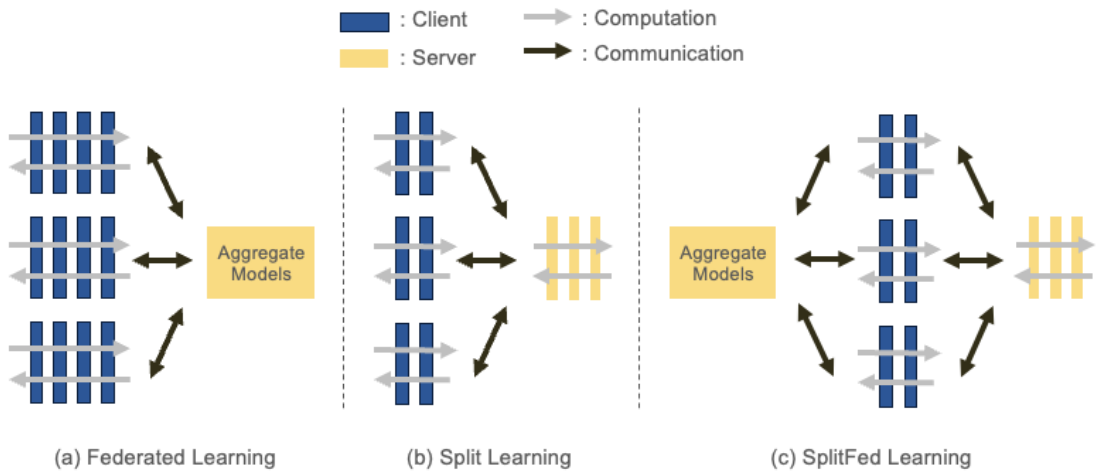


Figure 2. Comparison of FL, SL, and SFL: In FL, clients train full models; in SL/SFL, models are split between clients and server.

As a counterpart to training, edge inference refers to the execution of trained models directly on edge devices, where limitations in memory, computation, and energy affect performance. Various approaches have been proposed to make model inference feasible on edge devices, particularly in the areas of model design and compression. In terms of model design, researchers have developed both manually designed and automatically searched model architectures. MobileNets [32] use techniques such as deep-separable convolutions to construct lightweight deep neural networks, while NASNet [100] applies neural architecture search (NAS) to automatically discover efficient model structures. These approaches aim to find architectures that offer a good balance between accuracy and resource use for edge devices. Alongside architectural improvements, model compression techniques have also been widely explored to reduce the computational load and memory usage of models. Some of the common strategies include the low-rank approximation, which reduces parameter redundancy [38], and knowledge distillation, where a smaller model is trained to mimic the outputs of a larger model [8]. Compact layer design, as used in GoogLeNet [74], helps reduce ineffective computation within the network. In addition to these, network pruning has proven effective for shrinking model size and improving efficiency by removing unimportant weights and parameter quantization.

2.2.2 Advantages and Applicability

DML's advantages align with the evolving requirements of Edge Intelligence and privacy-preserving computation. Since raw data remains on local devices, DML helps preserve privacy and aligns with data protection laws. Also, instead of transferring full datasets, only model updates or intermediate features are exchanged [57], reducing the communication overhead. This is an advantage in rural, mobile, or bandwidth-constrained deployments. A promising direction for DML is to enable collaborative training on private, decentralized datasets that are currently underutilized. As high-quality public data for training large models becomes increasingly scarce, leveraging siloed but rich private data becomes essential for future scalability [95]. This includes valuable data, such as clinical records, financial logs, or proprietary sensor data, that cannot be centrally aggregated due to privacy or regulatory constraints. Federated fine-tuning thus offers a path toward scalable and privacy-preserving customization of large models. In addition, DML frameworks support asynchronous participation, allowing devices to join or leave the training process without disrupting global convergence [89]. This flexibility is essential in dynamic edge scenarios where power, connectivity, and availability are unpredictable. These applicabilities are supported by modern frameworks such as Flower, PySyft, and others [97], which incorporate techniques like quantization and parameter-efficient tuning for training on low-resource devices. Emerging use cases include decentralized instruction tuning, value alignment, and behavioral modeling, demonstrating DML's potential to support more inclusive, personalized, and secure AI development at the edge.

2.2.3 Technical and Practical Challenges

Despite the promise of DML, several technical and practical challenges remain that limit its widespread deployment and efficiency. These issues span data heterogeneity, coordination overhead, security vulnerabilities, and scalability barriers. The following are key challenges, each with specific implications for real-world applicability.

Straggler clients and non-IID data distributions: In DML, client devices often possess data that is *non-independent and identically distributed (non-IID)*, reflecting user-specific behaviors, environments, or tasks. This statistical heterogeneity can lead to different local updates from the global objective, resulting in slower convergence rates and reducing global model generalization. Moreover, some devices may compute updates more slowly due to limited resources or intermittent connectivity, creating *straggler effects* that delay synchronization and degrade overall training performance.

Synchronization overhead and communication latency: Coordinating model updates across a distributed network of heterogeneous devices introduces significant synchronization overhead. This challenge worsens performance in settings where devices have varying bandwidth, processing capabilities, or connection stability. High communication

latency, particularly in wide-area or mobile networks, can stall global model aggregation, leading to inconsistent or stale updates. These factors hinder real-time responsiveness and raise concerns about the practicality of synchronous DML at scale.

Adversarial threats and integrity compromise: The decentralized and often untrusted nature of DML participants makes the system vulnerable to adversarial attacks, including model poisoning (where malicious clients inject harmful updates) and data poisoning (where local training data are manipulated to skew learning outcomes). Such threats can compromise model integrity and trustworthiness, particularly in security-sensitive domains. Designing robust defense mechanisms such as anomaly detection, Byzantine resilient aggregation, and differential privacy remains an active and open area of research.

Scalability and aggregation bottlenecks: As DML frameworks scale to millions of devices, the need for lightweight protocols becomes critical. Traditional aggregation methods, such as centralized parameter servers or synchronous model averaging, do not scale efficiently and introduce communication bottlenecks and single points of failure. In order to ensure scalability, systems must employ decentralized aggregation schemes, hierarchical topologies, or eventual consistency models that reduce the dependency on centralized coordination while maintaining model accuracy and fairness.

2.2.4 Enhancement and Defense Mechanism in DML

While DML offers privacy-preserving training, its distributed nature and lack of centralized oversight expose it to various security threats, including poisoning, inference attacks, and unreliable participants [83]. To address these challenges without compromising performance, a range of defense mechanisms have been developed, particularly within paradigms such as FL and SL. The key mechanisms include the following.

Update-level defense techniques: In privacy-preserving DML settings such as FL and SL, where raw data remains local, many defense strategies operate on shared model updates instead. Robust aggregation methods, such as Krum [6] filter out anomalous updates to protect against untargeted attacks. Other approaches analyze intermediate activations or latent patterns to detect poisoned behavior, while techniques such as Neural Attention Distillation (NAD) [44] aim to erase malicious effects through knowledge distillation.

Differential privacy and secure aggregation: In privacy-preserving DML, mechanisms such as differential privacy add carefully calibrated noise to local updates before transmission, providing formal guarantees that individual data points cannot be inferred. While differential privacy may slightly reduce model accuracy, it significantly strengthens resistance against inference attacks. In tandem, secure aggregation techniques (e.g., homomorphic encryption or multi-party computation) ensure that the aggregator can compute the global update without ever seeing individual contributions in plaintext. This is critical in FL, where updates from thousands of devices are aggregated, and in SL,

where intermediate activations must remain confidential.

Reputation-based and blockchain-backed trust models: One emerging solution to mitigate poisoning and detect untrustworthy participants involves reputation systems that dynamically evaluate and score client behavior over time. Clients that consistently submit useful and stable updates are rewarded with higher trust, while suspicious clients can be penalized or excluded. In addition, blockchain-backed ledgers offer immutable audit trails to update provenance and verification, enhancing trust in fully decentralized environments. These approaches are especially relevant in DML settings with limited central oversight, such as edge-cloud federations or peer-to-peer networks.

Decentralized aggregation and gossip protocols: In order to reduce reliance on a central aggregator, decentralized aggregation schemes have been proposed [34]. These include gossip-based protocols, where clients exchange and average updates with a subset of their neighbors. Such mechanisms improve fault tolerance and robustness to adversarial clients, as no single entity has full control over model coordination. In FL, gossip protocols enable model updates to propagate in a more resilient and scalable fashion. In SL, while aggregation is less of a focus, peer-assisted architectures are emerging for shared layer training and redundancy.

Hybrid FL and SL architectures: Combining the advantages of FL and SL is gaining traction as a way to balance privacy, communication efficiency, and scalability. In these hybrid models, client devices perform initial training layers locally (as in SL), while later layers are updated via federated aggregation (as in FL). This reduces the burden on low-power devices and allows for task-specific model personalization. Additionally, FL/SL hybrids can isolate the impact of malicious clients by segmenting the training pipeline, making it harder for a single compromised device to poison the entire model. These hybrid schemes show promise for resource-constrained and privacy-critical scenarios such as smart healthcare, industrial IoT, and mobile personalization.

Taken together, these defense mechanisms provide a layered approach to enhancing the robustness of DML. However, many of them introduce tradeoffs: differential privacy may degrade accuracy, secure aggregation increases computation, and reputation models require long-term observation. Moreover, adversaries can adapt to defenses by crafting coordinated attacks that bypass simple filtering or masking strategies. As such, continued innovation is needed, particularly in adaptive attack-aware learning protocols that can dynamically detect, mitigate, and recover threats across large-scale untrusted networks.

2.2.5 Device Performance Metrics for Anomaly Detection

While many defenses in FL and SL rely on analyzing model updates or gradients to detect poisoning, these methods typically require access to internal model states, synchronized participation, or a trusted central aggregator. However, such assumptions are often unrealistic in edge deployments, where devices operate autonomously under privacy

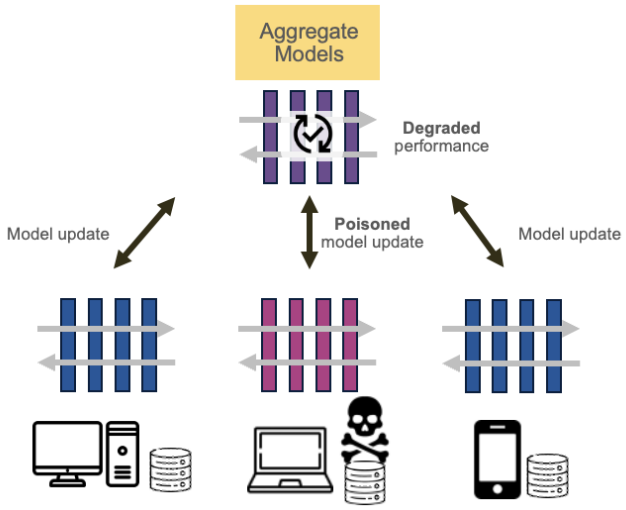


Figure 3. Illustration of a data poisoning attack in a DML setting. Compromised device trains on poisoned data and submits a manipulated update. When aggregated into the global model, this poisoned contribution degrades overall model performance.

constraints and unreliable connectivity. In order to address this challenge, a possible detection strategy involves monitoring the system-level performance behavior of devices such as CPU usage, temperature, memory load, or disk activity. These metrics are accessible, lightweight, and do not require knowledge of the model or training data. This approach has shown practical effectiveness in other security domains, motivating its potential application to the detection of poisoning in FL and SL and other collaborative DML. One example is the detection of denial-of-service (DoS) and distributed DoS (DDoS) attacks, which aim to overwhelm system resources by sending an excessive volume of requests, rendering the target unresponsive. These attacks cause noticeable spikes in system activity such as CPU load, temperature, and network throughput. For example, in UAV networks, researchers have developed detection methods based on features such as CPU utilization, device temperature, and packet transmission rates, achieving over 99% accuracy using ML classifiers [78]. Another example is behavior-based malware detection, where malicious applications are identified not by their code signatures but by deviations in runtime behavior. The Andromaly framework [68] for Android devices monitors system-level metrics such as CPU usage, battery consumption, and network activity to detect anomalies caused by malware. ThermWare [27] pushes this concept further, showing that thermal patterns on system-on-chip (SoC) hardware can reveal underlying computation behavior, allowing passive detection of malicious activity in micro-scale IoT devices. Building on these insights, system-level performance metrics monitoring could be a solution that runs during training based on the hypothesis

that malicious clients will exhibit anomalous system usage patterns.

2.3 Solutions for Automating Decentralized Systems Setup

2.3.1 Existing Automation Approaches and Limitations

As discussed in Section 2.1.3, edge intelligence deployments in real-world environments are hindered by challenges such as device heterogeneity, intermittent connectivity, rigid deployment models, and unstable networks. In order to address these issues, a variety of platforms and frameworks have been developed to automate edge system setup and simplify development and management. In the field of edge computing and intelligence, some open-source edge computing frameworks such as Azure IoT Edge [52], EdgeX, and KubeEdge simplify the integration of edge devices with cloud infrastructure. These platforms provide centralized control panels, protocol translation, and basic automation pipelines, often allowing users to deploy pre-trained ML models for inference at the edge. For example, Azure IoT Edge allows developers to install a lightweight runtime on edge devices, enabling them to run Docker-compatible modules directly at the edge. Devices are registered and authenticated through their Hub application, which serves as a centralized cloud interface for remote deployment, monitoring, and management. While these platforms help reduce engineering overhead, they are primarily designed for pre-defined infrastructures and fixed workflows.

Beyond edge-cloud platforms, P2P systems are also a decentralized collaboration. In P2P networks, the nodes interact directly to share data, perform computation, or coordinate tasks. The setup and management of such systems is usually done with simplified tools, which provide modular networking stacks for peer discovery, secure communication, and NAT traversal. Distributed storage systems automate content addressing and replication but still require manual configuration of nodes and bootstrapping peers. Similarly, decentralized file synchronization systems rely on user-defined rules and static peer relationships.

In the context of edge device selection and orchestration, researchers have explored heuristic-based automation methods to support the formation of dynamic infrastructure. These approaches typically address four stages: discovery, benchmarking, load balancing, and task placement [30]. Discovery protocols identify candidate edge nodes using techniques such as handshake protocols or message passing. Benchmarking methods assess resource performance, such as CPU, memory, and power, by lightweight profiling tools. Load balancing algorithms aim to distribute tasks across nodes using methods such as particle swarm optimization, cooperative load sharing, or graph-based repartitioning to avoid bottlenecks and improve Quality of Service (QoS) [80]. Placement strategies assign workloads based on resource availability, network conditions, and user location, using iterative or condition-aware techniques to meet application requirements and optimize execution. Despite this progress, current approaches face several limitations. Discovery

protocols often assume that edge nodes are publicly known or are part of a single administrative domain. Real-time benchmarking remains challenging due to the resource constraints of edge devices and the overhead of capturing accurate performance data on demand. These constraints limit the adaptability of existing orchestration techniques and highlight the need for more dynamic and context-aware solutions that can operate effectively in heterogeneous and resource-constrained edge environments.

2.3.2 Capabilities of Large Language Models

As traditional automation methods for decentralized infrastructures face increasing limitations in flexibility and adaptability, recent approaches have turned to LLMs to enable intent-driven orchestration and intelligent coordination. LLMs such as GPT and LLaMa are built on transformer architectures and are trained on massive amounts of text using self-supervised learning objectives [61]. Rather than relying on labeled datasets for specific tasks, LLMs learn to predict and model language patterns by processing large corpora of unstructured text. This allows them to acquire general linguistic knowledge, such as grammar, semantics, and world facts, during pre-training [63]. As models scale in size and training data, they begin to exhibit surprising generalization capabilities, including understanding complex instructions and reasoning about unfamiliar inputs [85]. These emergent behaviors are not explicitly programmed but arise from the sheer scale of model parameters and data exposure, making LLMs a powerful foundation for downstream applications. In order to adapt pre-trained LLMs to specific tasks without retraining, prompt engineering has emerged as a practical strategy. In this approach, users write input queries that are called prompts, which guide the model's output by providing examples, instructions, or context. Two common methods are zero-shot prompting [62], where the model receives only a task description, and few-shot prompting [7], where a handful of examples are included in the prompt to demonstrate the desired behavior. These techniques leverage the in-context learning ability of the model, allowing it to generalize across tasks with minimal additional supervision [64]. More advanced prompting strategies, such as chain-of-thought prompting, have also been shown to improve performance on reasoning tasks by encouraging the model to generate intermediate steps before reaching a final answer [86]. Prompt engineering plays a critical role in unlocking the potential of LLMs for flexible and domain-adaptive use.

2.3.3 The Role of AI and LLMs in Decentralized Infrastructures

The integration of AI and LLMs in decentralized infrastructure design is rapidly changing how intelligent edge systems are configured, deployed, and maintained. These technologies enable the adaptive management of infrastructure. Modern AI orchestration platforms now leverage LLMs to directly interface with infrastructure agility, efficiency,

and scalability [59]. This automation is particularly beneficial in heterogeneous edge environments, where low-level configurations vary significantly between devices, platforms, and vendors. Through prompt engineering, LLMs are capable of interpreting natural language, such as *"Optimize bandwidth usage for video analytics"* or *"Deploy a secure sensor cluster"*. These intents are then translated into executable workflows that reconfigure devices, enforce network policies, or allocate compute resources in real time [29]. AI can also be integrated into these systems to optimize data labeling workflows by identifying and relabeling when data are noisy, improving model performance while reducing annotation costs [31].

One of the most impactful capabilities of LLM-based agents is their ability to act as context-aware infrastructure advisors. By embedding domain-specific knowledge, such as network topologies, device capabilities, and task performance metrics, LLMs can continuously monitor system behavior and proactively recommend adaptive reconfigurations in response to changing workloads or partial failures [56]. LLMOps frameworks support this by managing model versioning, function execution, and monitoring in multi-tenant environments [17]. It ensures efficient use of compute resources and maintains inference latency within acceptable bounds. For example, an LLM-based agent can detect sustained packet loss at a remote gateway and autonomously propose a new routing strategy or a shift in workload to a nearby edge node. These agents also support human-in-the-loop control, allowing administrators to override, refine, or audit decisions made by AI systems, thus enhancing transparency and trust. In general, the convergence of AI, LLMs, and decentralized infrastructure management is not only reducing operational complexity but also enabling autonomous and resilient orchestration across dynamic large-scale edge ecosystems. This emerging paradigm is laying the foundation for next-generation AI systems that are scalable, context-aware, and continuously self-optimizing.

2.3.4 AI-driven System-Level Reasoning

LLMs can reason over device metadata, resource constraints, and network structure to propose task placements, resource assignments, or routing changes that align with the overall goals of the system. Techniques such as chain-of-thought prompting help externalize the LLM's reasoning steps, providing a transparent view into the model's decision process [86]. This fosters greater trust, interpretability, and degradability, which are essential attributes for mission-critical orchestration tasks. In real-world deployments, this can help operators understand why a model selected a specific configuration or rejected an alternative. Moreover, retrieval-augmented generation (RAG) frameworks improve LLM reasoning by integrating live telemetry data, infrastructure status, and technical documentation into the generation loop [26]. This allows the LLM to ground its recommendations in an up-to-date context, allowing more accurate and adaptive responses to system drift, failures, or usage spikes [19]. Despite these advances, one key challenge is keeping the LLM decision aligned with the actual state of the system as it

changes in real time. In dynamic environments, the drift between the model's internal assumptions and the actual infrastructure conditions can lead to incorrect or suboptimal recommendations [29]. Addressing this requires tighter coupling between LLMs and observability pipelines, as well as real-time validation and rollback mechanisms.

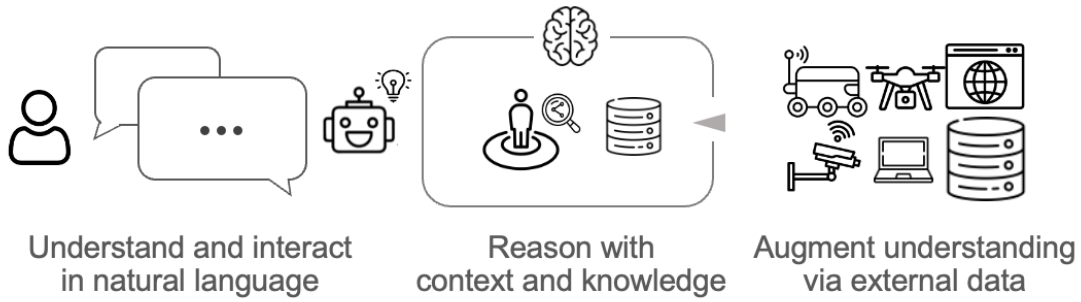


Figure 4. Core capabilities of LLM-based AI-Chatbots

2.3.5 AI-Chatbots for Decentralized Infrastructure Setup

Building on the system-level reasoning capabilities of LLMs, AI-Chatbots provide a conversational interface that helps non-expert users to interact with complex instructions using natural language. Recent research highlights a growing interest in it. Studies have explored their role in assisting users across diverse domains by optimizing conversational interfaces and enhancing the overall interaction experience [23]. An emerging area is negotiation, for which chat-enabled agents have been developed to handle multi-turn dialogues involving trade-offs, preference alignment, and dynamic strategy shifts [5]. These capabilities position LLM-based chatbots as collaborative agents that can support planning, reasoning, and decision-making alongside humans. Moreover, recent efforts have investigated the use of LLMs for supporting coordination across multi-device and edge environments. For example, pervasive chatbot frameworks are being explored to mediate interactions across multiple devices simultaneously [58], while other work examines how cloud-hosted LLMs can be leveraged to form lightweight, adaptive EdgeAI workflows [18]. These systems primarily focus on user interaction or proof-of-concept automation. They don't evaluate how LLMs perform in selecting or orchestrating infrastructure components based on the nature of the task at hand, which is a key aspect this thesis addresses. Recent applications such as HuggingGPT [69] and LLMind [15] demonstrate how LLMs can coordinate complex, multi-component tasks through natural language. HuggingGPT connects LLMs with AI models hosted on the platform, Hugging Face. It processes user instructions in four stages: task planning, model selection, execution, and response integration. By interpreting model descriptions

as language prompts, HuggingGPT enables the LLM to act as a controller that manages which external tools to invoke and how to sequence them. LLMind applies LLMs to the IoT domain, allowing users to issue high-level instructions that are translated into coordinated device actions. The LLM serves as an intermediary, interpreting user intent, identifying appropriate IoT devices, and enabling them to collaborate in real time. This approach eliminates the need for hard-coded control logic and allows the system to dynamically adapt to varying tasks and resources. These examples highlight the potential of LLMs to serve as intent-aware orchestrators, potentially for decentralized infrastructure settings where edge devices and services must coordinate flexibly and intelligently.

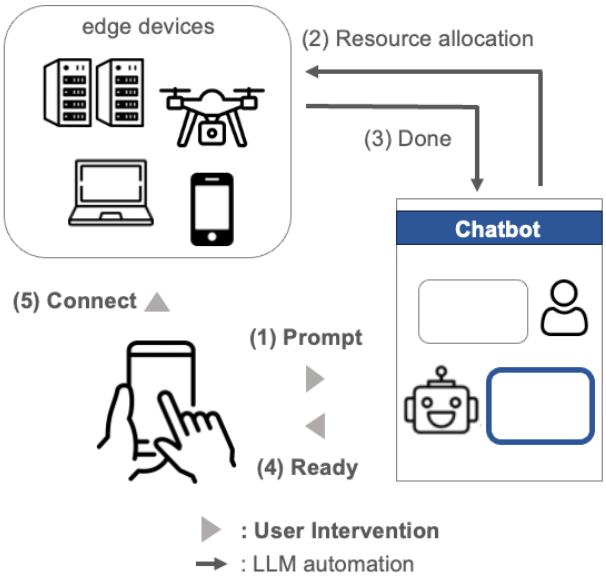


Figure 5. AI-Chatbot automation of decentralized system setup

2.3.6 Limitations and Open Problems

Despite their promising capabilities, LLM-based AI-Chatbots face several challenges when applied to the orchestration of decentralized systems. First, LLMs are computationally intensive, making them difficult to deploy on resource-constrained edge devices. Their high memory and processing requirements limit real-time use unless lightweight variants or cloud-based access are employed [88]. Second, LLMs are prone to hallucination, meaning they sometimes generate responses that sound plausible but are actually incorrect. This can pose serious risks in automation and system control scenarios [33]. Misinterpretations of user intent or inaccurate command translations can lead to

unintended behaviors across devices. Third, privacy and security concerns arise from the typical reliance on cloud-based inference. User data may need to be transmitted to remote servers, raising the risk of data leakage or misuse. Additionally, studies have shown that LLMs can be vulnerable to prompt injection or jailbreak attacks that bypass safety filters and trigger undesired actions [11]. These limitations highlight the need for careful design, robust safeguards, and possibly hybrid architectures that combine LLM-based flexibility with conventional reliability and verification mechanisms.

2.4 Summary

Decentralized systems distribute computing, storage, and control across many independent devices. Edge Intelligence builds on this idea by bringing artificial intelligence directly to edge devices. By processing data locally, these systems reduce latency, preserve privacy, and improve bandwidth efficiency. They also improve data sovereignty and system resilience in environments with unreliable or limited connectivity. However, building and managing these systems still remains challenging due to the heterogeneity of edge devices, varying operating environments, and network conditions. Several automation tools have been developed to help deploy and manage decentralized systems, often using scripts, visual interfaces, or container orchestration frameworks to mitigate those complexities. However, most existing setups are rigid, custom-built solutions, so they don't work well in dynamic opportunistic settings. DML allows devices to work together to train AI models by sharing model updates instead of raw data. These methods reduce privacy risks and communications costs, making them well-suited for privacy-sensitive domains and bandwidth-constrained environments. However, it is vulnerable to attacks such as data poisoning, where bad clients try to corrupt the model by sending harmful updates. Recent defense mechanisms typically assume full access to the training code and model structure, which is not scalable to heterogeneous devices in the real world. Despite recent advances, several critical gaps remain. First, the orchestration of edge infrastructures continues to rely heavily on manual configuration and expert oversight, limiting scalability and accessibility. Second, existing defense mechanisms for DML are often computationally intensive or dependent on centralized coordination, making them ill-suited for heterogeneous and resource-constrained edge environments. These limitations pose significant barriers to real-world adoption. In order to address them, this thesis investigates the application of LLMs for user-centric, automated infrastructure management and introduces a lightweight, model-agnostic approach for detecting poisoned clients based on observable device behavior metrics.

From this, we derive the following research questions:

1. How can LLMs be used to recommend and assemble optimal edge infrastructure configurations based on user-defined tasks and device capabilities?

2. Can open-source or locally hosted LLMs provide reliable support for infrastructure formation across diverse decentralized environments?

3. How do DML paradigms manage AI training and inference when exposed to vulnerable devices, particularly in FL and SL settings during monitoring performance, and can the resulting runtime effects be detected using lightweight, decentralized metrics?

4. Can device-level performance signals, such as CPU usage and execution time, be used to accurately detect vulnerable clients in distributed learning without accessing the models or training data?

3 Motivation: Optimal Edge Infrastructure Assembly Based on LLM

Recent advances in Edge Intelligence and DML have led to a wide range of applications, from real-time AI inference to collaborative model training across edge devices. However, practical deployment and management of these systems remain challenging in dynamic, resource-constrained environments. Existing frameworks often rely on static, preconfigured infrastructures, which introduce technical overhead and reduce adaptability. In order to simplify deployment, many are implemented as specialized clients with fixed functions. While this reduces complexity, it constrains flexibility, requires manual intervention, and hinders responsiveness to changing workloads or device availability.

In this chapter, we explore the potential of LLMs as intelligent agents for user-centric and adaptive orchestration of edge infrastructures as used by [39]. Specifically, we examine whether LLMs can reason over device specifications and task requirements to support user-friendly system formation through natural language interaction. In order to investigate this potential, we conducted two exploratory experiments. The first evaluates how GPT-based models (GPT-4o and GPT-3.5 Turbo) recommend suitable edge devices for various task categories, such as high-performance computing and sensor-based data collection. The second examines the baseline reasoning capabilities and deployment feasibility of open-source LLMs, focusing on LLaMa 2 models hosted in resource-constrained environments.

LLM selections: The selection of **GPT-4o** and **GPT-3.5-Turbo** for the first experiment is driven by their state-of-the-art performance in natural language understanding, reasoning, and few-shot generalization. As of when these experiments were conducted, these models represent the most advanced offerings in OpenAI's GPT series. *GPT-4o* is optimized for faster inference and broader multimodal input handling. *GPT-3.5-Turbo*, though lighter and more accessible, retains strong performance, making it suitable for evaluating the baseline capabilities of LLMs in task-aware infrastructure planning. For the second experiment, we selected **LLaMa-2-7B-Chat**, an open-source LLM family developed by Meta, to evaluate the feasibility of deploying LLMs in resource-constrained or locally hosted edge environments. The 7B parameter variant was chosen because of its balance between reasoning ability and inference efficiency, allowing it to be hosted on a single GPU or a lightweight server setup. Additionally, the open-source nature of LLaMa 2 allows greater customization, fine-tuning, and offline operation. Including this model provides information on the scalability and accessibility of the proposed orchestration method, especially in scenarios where relying on commercial APIs or cloud infrastructure is impractical.

Together, these experiments provide early insight into how LLMs can support task-aware device selection and natural language-guided orchestration in edge settings. The findings motivate the integration of LLMs in decentralized infrastructure management,

forming the basis for the methodology proposed in the next chapter.

3.1 Experiments

3.1.1 LLM Device Recommendation for a Task

We first constructed a dataset of more than 200 smartphone models released between 2020 and 2024. While the system is designed to coordinate a wide range of opportunistic edge resources, including drones, laptops, and IoT devices, this experiment narrows the scope to smartphones, which serve as a representative class of edge devices due to their ubiquity, heterogeneous hardware capabilities, and integrated sensors. These characteristics make smartphones a practical and diverse testbed for evaluating LLM-based infrastructure assembly. The dataset includes a wide range of devices, from entry-level phones to flagship models. For each device, we collected three benchmark attributes: CPU and GPU scores from Geekbench ML results,[1], and camera quality scores from DxOMark evaluations.[2] Notably, camera quality is not accurately represented by resolution alone. The DxOMark's scoring methodology accounts for complex imaging factors including exposure accuracy, dynamic range, color rendering, detail preservation, low light performance, and noise handling, making it a robust ground-truth reference. These benchmark data were used exclusively for later evaluation and were not exposed to the LLM.

The experiment simulates three types of edge tasks related to ML, each focusing on a different hardware requirement: CPU, GPU, or camera. These task categories are designed to reflect realistic deployment scenarios in fields such as public safety, smart agriculture, and environmental monitoring. CPU-intensive tasks include local control algorithms or data aggregation and preprocessing; GPU-intensive tasks include image classification, visual rendering, or AR-related workloads. For tasks that depend on visual input, such as image-based sensing, object recognition, or mobile video analytics, high camera quality is essential. For each task, 20 smartphone model names were randomly sampled from the full dataset to represent nearby candidate devices. The LLM was prompted to select the five most suitable devices for the given task. These interactions were executed via the OpenAI API using two models: GPT-4o and GPT-3.5-Turbo. Each session consisted of a system prompt and a user prompt. The system prompt defined the assistant's role as a technical helper for distributed edge systems, while the user prompt included the task description and the list of candidate devices. Full examples of these prompts are provided in Table 1. In order to avoid any bias from memory across runs, each prompt was executed in a fresh, stateless session. Each task configuration was repeated five times to assess the consistency of the recommendation.

[1] `https://browser.geekbench.com/`

[2] `https://www.dxomark.com/smartphone-camera-image-quality-test/`

Role	Features	Prompts
System	-	You are a helpful assistant for creating distributed systems. You'll read smartphone device data and answer questions about it. Since the questions might not always be straightforward, so you'll start by figuring out what they're trying to do and which features of the smartphone are most critical for that task.
User	CPU	Hey, I want to analyze pictures as fast as possible. Can you quickly help me with 5 devices nearby that can help with this?
User	GPU	Hey, I want to make an app to recognize animals from pictures, and I want to use the fastest devices to split the work. Select 5 devices.
User	Camera	Hey, I want to get the best quality pictures. What 5 devices should we use to get the best images?

Table 1. Prompts for assembling decentralized devices

Scoring methodology: In order to evaluate the quality of the LLM-generated recommendations, we manually compared the selected devices with benchmark-based baselines. For each LLM response, we retrieve the corresponding benchmark scores from the dataset and calculate the total score by summing the relevant performance metrics across the five selected devices. For CPU-focused tasks, we summed the CPU benchmark scores; for GPU-focused tasks, the total GPU benchmark score was used; and for camera-centric tasks, we calculated the sum of the DxOMark camera scores. These totals were compared with two baselines:

1. **Top-5 Baseline:** This is the total benchmark score of the five highest scoring devices for the target metric (CPU, GPU, or camera) within the 20 candidate devices.

2. **Random Baseline:** This is the expected total score of five randomly selected devices, calculated by computing the average score of the 20 candidate devices and multiplying it by 5.

For the camera task, we added an additional Resolution-Based baseline using the top five smartphones ranked by camera resolution. This was included to demonstrate that resolution alone is not a reliable proxy for sensing quality. The performance of each selection was averaged over the five repetitions per model and task. This allowed us to assess both the accuracy and stability of the LLM's internal reasoning and its alignment with real-world device capabilities.

3.1.2 Practical Knowledge in Open-Source LLMs

The experiment used the LLaMA 2-7B-Chat model [3], released by Meta at HuggingFace, which was chosen for its balance between model size and performance. In order to run on devices with limited resources, the model was converted into the GGUF (GPT-Generated Unified Format) format. GGUF is a lightweight binary format designed for efficient on-device inference. It consolidates the model weights and metadata into a single compact file and supports quantization, which significantly reduces memory requirements while maintaining the model's core functionality. This format is particularly useful for running large models on machines without high-end GPUs. The model was served using the llama-cpp backend, an open-source inference engine optimized for LLaMA-based models. llama-cpp is implemented in C++ and supports both CPU-only and GPU-accelerated execution, with minimal dependencies. It is known for its low memory footprint and fast performance, making it well-suited for deployment on edge or embedded systems. Despite its lightweight nature, it supports full chat functionality, including multi-turn dialogue and role-based prompts. The deployed model was hosted on a cloud instance, which in this experiment refers to a virtual machine running on a local server rather than a commercial cloud provider. Specifically, the instance ran Ubuntu 22.04 with 4 virtual CPUs, 8 GB of RAM, and 25 GB of storage. This virtual machine was created and managed using virtualization tools on a standard laptop, simulating a private edge cloud setup. The LLM was exposed via a REST API using FastAPI, a high performance Python web framework. This setup emulates how an LLM-based chatbot would be integrated into a decentralized system, allowing edge clients (such as smartphones) to communicate with the model via lightweight HTTP requests. In order to interact with the deployed LLaMA backend, we developed a lightweight Android application that acts as a front-end chatbot interface. The app allows users to enter natural language queries and sends them to the back-end via HTTP requests. The responses from the LLM are displayed in a chat interface on the device. This setup simulates a real-world usage scenario in which an edge device acts as the user-facing entry point for infrastructure orchestration. The chatbot was accessed via a Huawei P30 smartphone, which served as the user device in this experiment.

In this setting, the LLM was not connected to any external databases, online APIs, or structured specification repositories. It received basic device metadata collected locally from nearby devices using Bluetooth and Wi-Fi Direct. This local discovery process provided information such as device names, hardware model identifiers, and architecture, mimicking how opportunistic edge resources might be identified in a real deployment. The LLM then used its background knowledge acquired during pretraining to infer additional details about each device and respond to user queries accordingly. Users submitted a variety of questions relating to smartphones, drones, and IoT devices,

[3] lhttps://huggingface.co/meta-llama/Llama-2-7b-chat-hf

focusing on whether the model could provide useful hardware-related insights to guide infrastructure formation. These included factual queries, for example, *"Can you suggest DJI drones with a flight time of 25 mins or more?"* as well as more open-ended prompts such as *"Which device has better video quality?"*. Figure 8 in the next section shows the examples of full prompts.

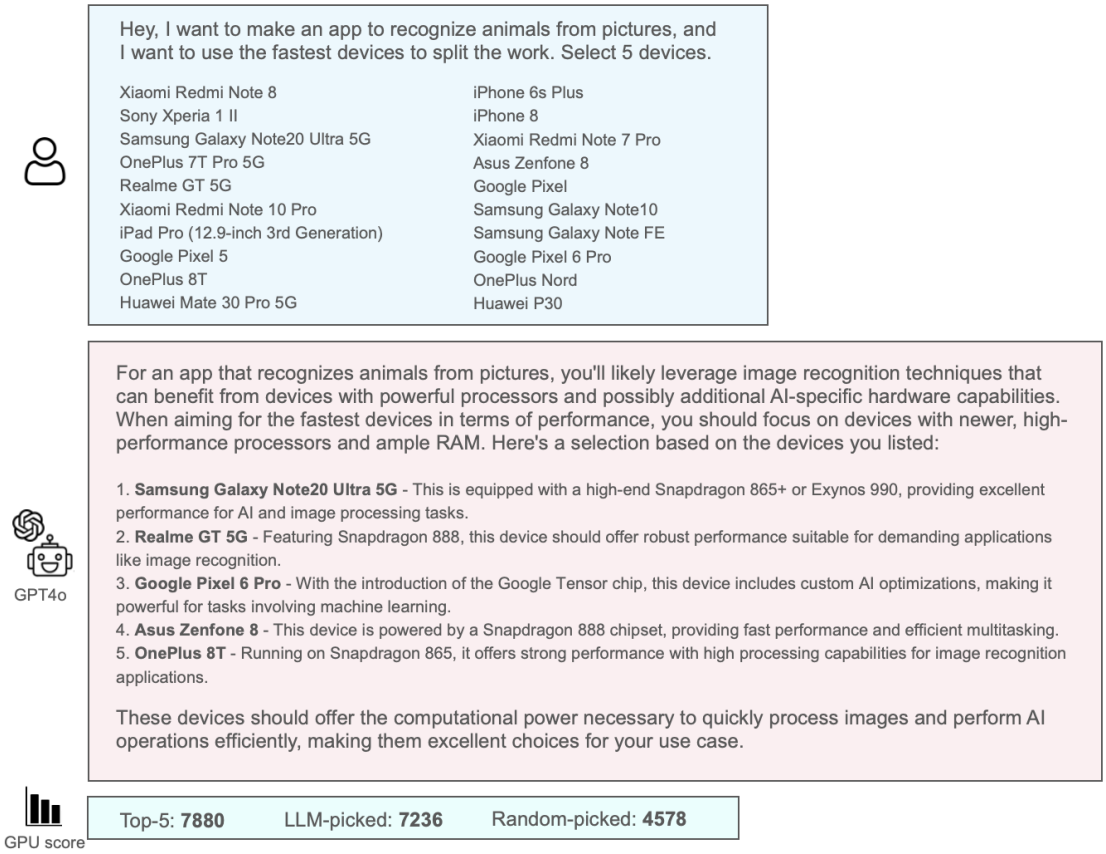


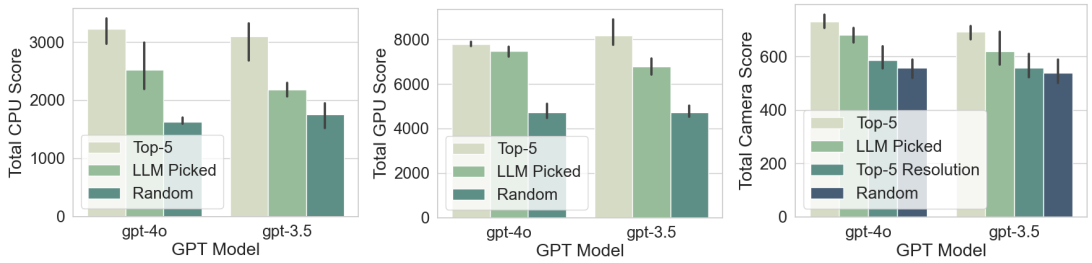
Figure 6. Example of a full GPT-4o prompt and response for smartphone selection in a task requiring a strong GPU.

3.2 Results

3.2.1 LLM Device Recommendation for a Task

Our Figure 7 summarizes the benchmark performance of devices selected by GPT-4o and GPT-3.5-Turbo across three types of tasks: CPU-intensive, GPU-heavy, and tasks that rely on visual input. Each bar chart compares the total benchmark score of the top-5 devices selected by the LLMs against two baselines: the actual top-5 devices in the 20 candidate

devices(upper bound) and five randomly selected devices. For the camera-based task, an additional baseline, top-5 devices based on their resolution value, was added. In both the CPU and GPU tasks (Figures 7a and 7b), LLM-selected configurations consistently outperformed the random baseline and approached the performance of the top-ranked devices. GPT-4o achieved up to 1.5× higher scores than random selection, demonstrating a strong alignment with benchmark-based ground truth, although the model was not provided with performance data. For the camera quality task (Figure 7c), GPT-4o demonstrated the ability to prioritize high-end camera phones, with recommendations closely matching the DxOMark rankings. The devices with the highest resolution were not among those with the best camera quality. The selection of LLMs more accurately reflected these complex trade-offs, suggesting a level of reasoning about sensing performance comparable to that of human experts. Figure 6 presents an example of the experiment log, showing the selection process from 20 candidate devices to the final 5. It includes the LLM's responses and reasoning steps, demonstrating how the model evaluates device attributes and matches them to the task requirements during orchestration. The results suggest that both GPT-4o and GPT-3.5-Turbo can effectively infer device suitability using only model names and task descriptions, without structured input. GPT-4o consistently performed better in accuracy and consistency, but GPT-3.5-Turbo still demonstrated a reasonable degree of performance awareness.



(a) Total CPU Score (b) Total GPU Score (c) Total Camera Quality Score

Figure 7. LLM-based device recommendation performance for edge task scenarios: (a) CPU-heavy task, (b) GPU-accelerated task, and (c) camera-based sensing. Bars compare GPT-selected devices with the actual top-5 performers, random selections, and (for cameras) top-5 resolution devices.

3.2.2 Practical Knowledge in Open-Source LLMs

Users submitted queries related to hardware attributes relevant to the edge in three types of infrastructure: autonomous drones, smartphones, and IoT devices (e.g., Raspberry Pi peripherals). The prompts were formulated to reflect realistic orchestration use cases, such as selecting drones with adequate flight time, identifying smartphones with specific

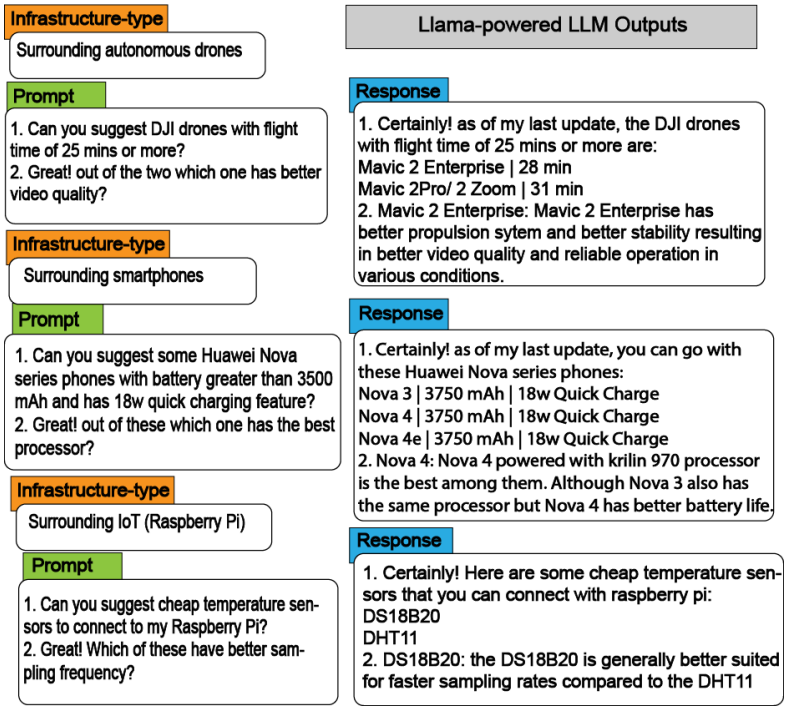


Figure 8. AI-Chatbot responses powered by LLaMA 2-7B-Chat. The chatbot infers device and sensor specifications across three infrastructure categories using only background knowledge.

battery and charging characteristics, or comparing sensors for edge monitoring. Figure 8 shows examples of the prompts and corresponding responses.

The results demonstrate that LLaMA was able to provide reasonably accurate information about well-known devices and components. For example, it correctly identified DJI drones with over 25 minutes of flight time, compared processor performance among Huawei Nova series smartphones, and distinguished the sampling rates of common Raspberry Pi temperature sensors. These responses indicate that the model has retained factual knowledge relevant to device capabilities, even without structured input. One notable strength is the model's ability to respond to follow-up prompts in a conversational format. In each case, the chatbot provided initial recommendations, followed by refined analysis when asked to choose the best option based on specific criteria (e.g., video quality, sampling rate, and processor strength). This supports the usability of the AI-Chatbot as a low-friction, dialogue-driven interface for edge system orchestration.

Foundations for edge intelligence copilots: Recent work, such as [60], has explored how LLMs can enrich sparse sensor data and act as interactive reasoning engines for users. Figure 9 demonstrates how AI-Chatbots with LLMs can be a foundational tool for orchestrating edge intelligence. Instead of requiring technical expertise to configure

devices or write scripts, users simply express goals through natural language, such as analyzing foot traffic, identifying lost items, or understanding customer behavior. The LLM interprets these instructions, selects suitable edge devices, and initiates collaborative operations. When needed, it can also prompt follow-up questions to clarify intent or adapt to changing contexts. These examples span mart city operations, public assistance, and smart planning, illustrating how this approach supports not just individuals but also local governments or enterprise operators. By lowering the barrier to coordination, it offers a practical and accessible way to manage decentralized, real-world systems.

Figure 9. AI-Chatbot coordinating Edge Intelligence tasks, adapting to user context and environment.

3.3 Summary

The results demonstrate that LLMs can effectively interpret user intent and reason about which performance factors are most relevant for a given task. Leveraging their broad background knowledge, they recommend optimal combinations of edge devices, even outperforming simple spec-based selections in certain cases. This capability forms the basis for the optimal and dynamic assembly of heterogeneous edge infrastructures. Open-source models such as LLaMA 2 also showed comparable reasoning abilities, supporting lightweight, locally hosted deployment. The conversational interface allows users to explore or refine their requests through natural follow-up prompts, making infrastructure planning accessible and intuitive. These findings motivate the design of the LLM-driven orchestration framework introduced in the following chapter.

4 Methodology

In this chapter, this thesis proposes *Edge Intelligence on Command*, a user-centric architecture that enables natural language-driven ML orchestration over opportunistic edge infrastructures. Motivated by the exploratory findings in the previous chapter, which demonstrated the potential of LLMs to reason over task requirements and device capabilities, the system helps users to specify ML tasks using natural language, which the LLM then interprets to discover appropriate edge devices capable of training or inference workflows. This user-centric management removes the need for low-level configuration that is often associated with the dynamic nature of edge environments.

However, effective orchestration in DML involves more than just matching tasks to devices. At the edge, learning often depends on contributions from heterogeneous and mobile systems, such as autonomous drones, service robots, and embedded sensors. In such settings, corrupted data may result not only from malicious actors but also from environmental interference or hardware degradation. For example, a drone operating in an urban setting might suffer from a smeared or obstructed camera lens, either by accident or external tampering, resulting in blurry, occluded images that produce misleading or degraded training data. Even in the absence of malicious intent, such contributions can significantly impair the quality of the global model.

In order to ensure that collaboration across edge devices remains reliable, the proposed architecture incorporates a lightweight anomaly detection mechanism alongside the orchestration layer. Designed for decentralized, resource-constrained environments, this mechanism monitors device-level behavioral signals, such as execution time, resource usage, or task performance, to identify anomalies that may indicate corrupted or low-quality contributions. By detecting such issues early, whether caused by malicious interference or unintentional faults, the system preserves the integrity of distributed training and inference workflows. This chapter outlines the design goals and architectural components of the proposed system.

4.1 Design Goals

The proposed architecture is shaped by two primary objectives that reflect the central contributions of this thesis:

User-friendly edge orchestration: The first goal is an adaptive coordination of edge resources through natural language interfaces. At the core of this functionality is an LLM-powered AI-Chatbot, which parses user intent, reasons for task-specific requirements, and selects optimal devices in real time. Unlike static orchestration approaches, LLM uses chain-of-thought reasoning to weigh factors such as computational capacity, memory availability, battery life, and sensor quality. This reasoning-driven selection allows the system to support a wide variety of ML tasks across dynamic, heterogeneous edge

environments, without requiring pre-defined task-device mappings.

Lightweight, anomaly detection between edge components: The second goal is non-intrusive, distributed monitoring of training behavior to detect potential poisoning or corruption. Instead of relying on centralized inspection or computationally expensive anomaly detection, the system introduces a simple yet effective behavioral metric to flag potential anomalies during training. The selected approach is deemed to be decentralized, model-agnostic, and well-suited to edge settings with limited visibility, intermittent connectivity, and varying device capabilities.

4.2 System Overview

The system leverages the reasoning capabilities of LLMs to bridge the gap between non-expert users and complex system-level decisions traditionally handled by DevOps engineers or AI specialists. By abstracting technical details through an AI-Chatbot interface and integrating runtime behavioral monitoring, the system facilitates both intelligent device selection and lightweight anomaly detection. The proposed architecture is designed to operate under real-world edge conditions characterized by fluctuating device availability, heterogeneous hardware, and data locality constraints. Figure 10 illustrates the four core phases of the system workflow, from edge resource discovery to the final delivery of an AI model. Each phase is tightly integrated with the others and contributes to the overall objective of building a flexible, resilient, and adaptive edge learning platform.

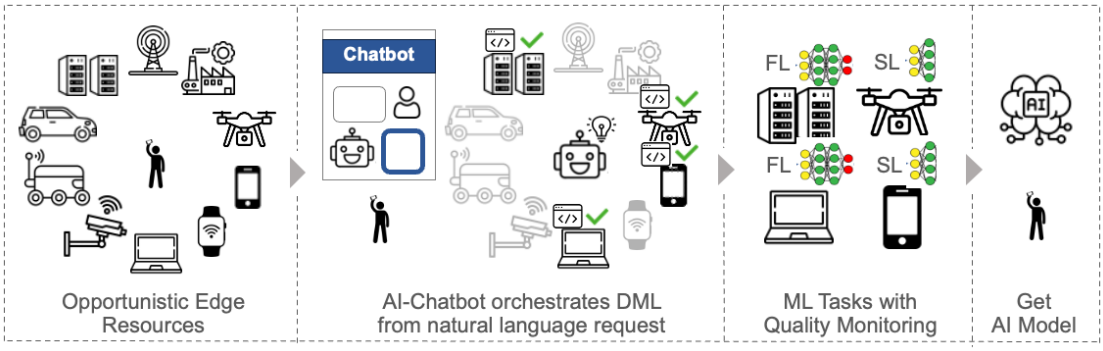


Figure 10. Edge Intelligence on Command — Workflow

Phase I - Availability of opportunistic edge resources: This phase initiates the system pipeline by identifying the available edge resources within a dynamic environment. Unlike static or preconfigured infrastructure typical of centralized data centers, edge environments are characterized by heterogeneous, mobile, and intermittently connected

devices. These include smartphones, tablets, laptops, embedded controllers, and autonomous platforms such as service robots or drones [20]. Such devices are typically not provisioned for dedicated training but can opportunistically contribute computation and data when idle or underutilized. The term opportunistic refers to the non-deterministic availability and reliability of these devices. Discovery mechanisms rely on proximity-based service discovery protocols (e.g., mDNS, Bluetooth Low Energy, or Wi-Fi Direct) or cloud-assisted registries that detect devices based on network presence and capability broadcasts [46]. Each device provides a resource descriptor that includes metadata such as processor architecture, memory size, battery state, sensor types, and security posture. The motivation behind this phase is to democratize access to distributed AI by leveraging edge heterogeneity as a strength, not a limitation. Using ad hoc networks of commodity devices, the system significantly reduces the barrier to deploying ML workloads, especially in environments with limited infrastructure or high mobility, such as disaster response, rural health monitoring, or smart agriculture [66].

Phase II – Task request and DML orchestration with natural language interaction: This phase transforms high-level user intents into executable distributed learning tasks through a natural language interface powered by a domain-specialized LLM chatbot. Inspired by intent-based paradigms but adapted for flexibility, the system allows non-expert users to express goals such as: *"Train a gesture recognition model using nearby smartphones."* or *"Classify audio input using distributed microphones."*. These intents are translated into structured task descriptors that include the type of operation, data modality, model preferences, latency [15], and privacy requirements [3]. The system then performs multiple criteria reasoning to select an optimal set of edge devices based on hardware capability, resource load, energy state, network quality, sensor relevance, and data availability. Rather than static filtering, LLM uses chain-of-thought prompting [86] and embedded domain knowledge for dynamic optimization, possibly incorporating historical performance or trust metrics [25]. Based on these selections, the orchestration engine configures and deploys the learning task between the chosen devices, such as FL and SL. This closes the loop from natural language input to live execution, replacing manual setup with AI-driven orchestration [58].

Phase III - ML task execution and performance monitoring: The system distributes the ML workload across opportunistic edge devices using collaborative learning strategies such as FL, SL, or local inference, depending on the deployment context and hardware capabilities. For example, both FL and SL paradigms are communication efficient and inherently privacy-preserving, making them suitable for edge environments. During task execution, the system continuously monitors performance indicators of each participating device, such as CPU usage, memory load, temperature, disk I/O, and training duration. In order to ensure that this monitoring framework remains lightweight and generalizable, its viability will first be assessed through a dedicated feasibility study. This study will explore whether changes in runtime metrics could reliably reflect underlying anoma-

lies such as model poisoning or degraded data quality, without introducing significant overhead. Additionally, based on these insights, the system will proceed to validate the distinct shifts by mapping them to a lightweight indicator of abnormal device behavior for efficient monitoring.

Phase IV - Result retrieval: The final phase involves the aggregation, evaluation, and return of the trained or updated model to the user. In FL settings, this includes model fusion techniques such as FedAvg or FedProx to merge local updates into a global model [43]. In SL settings, activations and gradients are stitched to reconstruct the full model state. The output is an AI model adapted to local context, trained collaboratively without centralizing data. The model may then be returned to the initiating user, deployed back to edge nodes to generate predictions, or logged for future fine-tuning. This phase may also invoke validation steps using holdout datasets or cross-validation on trusted nodes. Furthermore, user feedback can be looped into continuous learning cycles, creating an evolving system that self-improves through real-world use [79].

This chapter presented the design rationale, system architecture, and feasibility validation of our proposed framework. Together, these components establish a foundation for the dynamic, secure, and user-friendly deployment of AI tasks across decentralized infrastructures.

5 Experiment Setup

Building on the preceding chapter, we introduced our system methodology, outlining a four-phase approach: (I) user intent via natural language, (II) task orchestration and device selection, (III) collaborative training deployment, and (IV) runtime monitoring. This chapter turns to the core of our experimental evaluation, which focuses on Phases III and IV, specifically, runtime performance monitoring in DML environments. Once the system is live, new challenges emerge, particularly adversarial threats stemming from unverified, heterogeneous devices. To address these, we instrument each DML paradigm with lightweight monitoring probes that capture system-level metrics such as model accuracy and anomaly signals in real time. Using a widely recognized dataset, we assess how each paradigm performs under natural language-driven orchestration and extract insights for improving runtime resilience in edge deployments. We assume that Phases I and II are completed such that the AI-Chatbot orchestrator has already been fine-tuned for edge intelligence tasks. It not only interprets user intent and assembles appropriate infrastructure but also considers the training and inference quality. Based on contextual reasoning, it incorporates the proposed runtime monitoring system without requiring explicit user instructions, especially for adversarial attacks. In this scenario, the AI-Chatbot has already configured and deployed the edge infrastructure in response to the following user intent: *"Can we make a system that helps recognize road signs using local footage?"*

5.1 Feasibility Study for Anomaly Detection Using Device Behavior

One of the key objectives of an efficient DML architecture is to maintain the integrity of the learning process as heterogeneous devices dynamically join the network, contribute data, and participate in model updates. This section presents a feasibility study that investigates whether poisoned data can induce measurable changes in runtime behavior on edge devices. The study is inspired by the ThermAware approach [27], which demonstrated that data poisoning during training could lead to abnormal thermal patterns detectable by external thermal cameras. Our study tests this hypothesis in a representative edge computing setup, aiming to generalize the idea toward scalable, built-in behavioral metrics.

Apparatus: The experimental platform consists of a Raspberry Pi 4 Model B, equipped with a quad-core Cortex-A72 CPU and 8GB RAM. This device represents typical computing capabilities of mobile or autonomous edge agents, such as drones or service robots. In order to monitor physical behavior, a CAT S61 smartphone with a built-in FLIR thermal camera was used to record external temperature profiles during training.

Hypothesis: We hypothesize that data poisoning will affect not only the model performance but also the physical execution characteristics of the device, such as thermal output

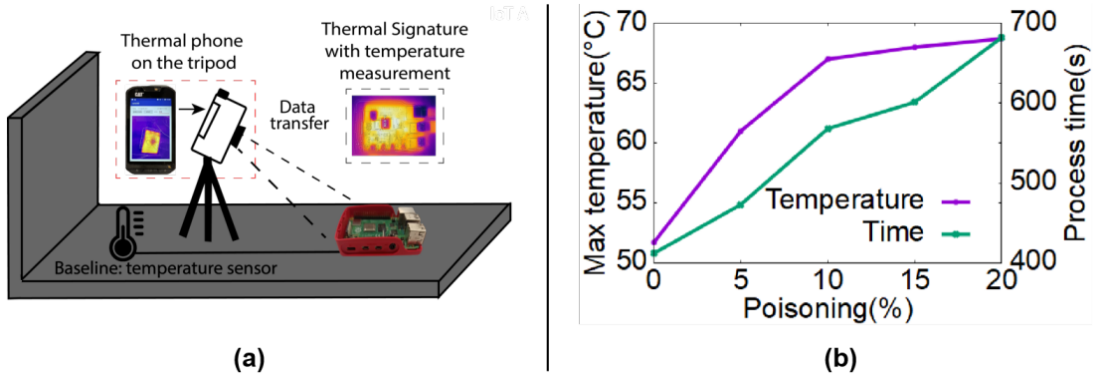


Figure 11. Detection of poisoning attacks: a) Experimental testbed, b) Insights using CPU temperature (Raspberry PI 4)

and resource usage. Specifically, we expect that increasing levels of poisoned data will lead to noisier gradient updates, slower convergence, and higher runtime temperatures.

Method: Two common data poisoning strategies were employed: *Label flipping*, where class labels are reassigned to incorrect categories, and *Occlusion*, where a portion of each image is randomly covered with a black patch. Each strategy was applied independently to the training dataset, with poisoning ratios of 5%, 10%, 15%, and 20%. For this experiment, we used the Chinese Traffic Sign Recognition Benchmark dataset[4], which reflects vision-based classification workloads commonly found in embedded edge systems such as smart vehicles or traffic infrastructure.

Procedure: With each poisoning level and strategy, the model was trained on the corresponding poisoned dataset using the Raspberry Pi device. Figure 11(a) shows the experiment setup. During the training period, a CAT S61 smartphone equipped with a FLIR thermal camera continuously recorded the surface temperature of the device. In parallel, internal metrics such as training duration and CPU usage were recorded to capture execution behavior. In order to establish a reliable comparison, control experiments using clean (unpoisoned) datasets were carried out under identical environmental conditions, allowing the impact of poisoning to be isolated and quantified against a consistent baseline.

Results: As shown in Figure 11(b), both the average CPU temperature and duration of training increased consistently with higher poisoning ratios. For example, under 20% occlusion, the temperature increased by an average of 5.3°C compared to the clean baseline. The training time also increased by 18 to 22% depending on the type of poisoning, confirming that convergence was altered. These results validate the hypothesis that poisoned data affects not only model learning but also detectable signatures in device

[4]https://nlpr.ia.ac.cn/pal/trafficdata/recognition.html

behavior.

Implications: While thermal sensing confirmed the feasibility of behavior-based poisoning detection, it remains impractical for real-world deployment due to its reliance on external sensors. This motivates a shift toward in-device telemetry, where metrics such as CPU usage, memory consumption, training duration, and onboard temperature logs are used to infer anomalies. The next section presents the overall system workflow and describes how a chosen metric can be integrated into run-time anomaly detection for decentralized learning settings.

5.2 Non-Intrusive Detection of Edge Device Anomaly

The motivation behind this evaluation is to detect behavioral anomalies during collaborative training in edge environments, without relying on centralized oversight or direct access to raw training data. When edge devices contribute corrupted or low-quality data due to environmental interference, user misuse, or system faults, these updates can degrade the quality of the global model. To mitigate this, the system must identify when a device's behavior deviates from expected patterns using only runtime signals, rather than inspecting model parameters or private data directly. Three experiments were conducted to explore this direction. The first investigates how poisoning strategies and intensities affect model performance in FL and SL settings, and whether such effects correlate with measurable runtime signals. This includes the introduction of the DCPI, a lightweight behavioral metric derived from signals such as CPU usage and training duration. The second experiment evaluates the robustness of DCPI-based detection under varying background processing conditions, while the third assesses its generalizability across different datasets, models, and learning setups. Rather than focusing on preventing attacks or ensuring security, these experiments aim to support adaptive responses to unreliable device contributions. By monitoring fluctuations in runtime behavior, the system can detect anomalies without centralized coordination or access to sensitive raw data, leading to practical on-device anomaly detection for collaborative edge learning.

5.3 Adversary Model and Assumptions

We define the threat model as a description of the attacker's goals, capabilities, and limitations that guide our experimental setup to evaluate data poisoning in DML systems. In general, adversarial attacks on ML models can be categorized into three types: exploratory, evasion, and poisoning attacks [24]. This work focuses specifically on data poisoning, in which compromised participants attempt to corrupt the training process by injecting misleading or manipulated input data. This form of attack degrades the collaboratively trained model by distorting the learning signal, even when the model architecture and training logic remain unchanged. We assume a *black-box poisoning scenario* in which the attacker has no access to the internal architecture, training parameters, or model

updates. Instead, the attacker participates in the DML process as a client and has full control over the local training data. This reflects real-world deployments of edge systems, such as autonomous drones, smartphones, or IoT sensors, where devices collect data from the physical environment and use it for local model updates. In such settings, an attacker can smear or block a camera lens, introduce physical occlusions, or embed misleading visual patterns in the environment to alter what the device perceives. Figure 12 illustrates this threat model. Each edge device participates in collaborative training by locally updating a shared global model. The poisoned device (top left) contributes tampered data to the training process, such as manipulated object labels or visually altered inputs, which influences the global model update and, in turn, affects the other clients. The other devices (bottom) perform standard learning based on local observations of real-world objects like trash or containers. This setting highlights the vulnerability introduced by one compromised participant in an otherwise cooperative learning environment. In order to evaluate this scenario, we simulate four types of poisoning attacks in our experiments: *Blurring*, *Label flipping*, *Occlusion*, *Steganography*. The details of these poisonings are shown in Section 5.6. Each of these types of attacks modifies only the input data; no changes are made to the learning algorithm, the model weights, or the software execution. We further assume that attackers cannot interfere with system-level monitoring or the measurement of run-time behavior. Performance metrics such as CPU usage, memory load, and CPU temperature are assumed to be logged independently of the training data and used later for anomaly detection. This setup reflects a minimal but realistic threat model, where only data-level corruption is possible, and no access to internal model logic or monitoring infrastructure is available.

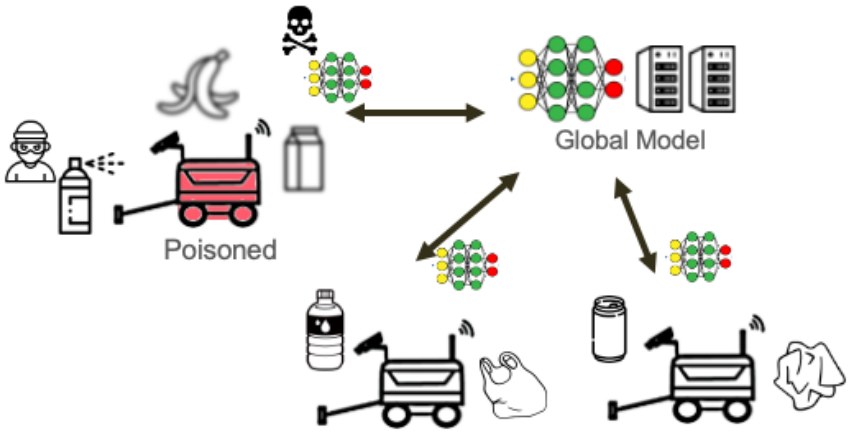


Figure 12. Assumed black-box data poisoning threat model. The attacker injects visually manipulated or mislabeled training data.

5.4 DCPI Calculation and Runtime Classification

In order to support poisoning detection without access to training data or model weights, we introduce the DCPI, a lightweight, hardware-agnostic metric derived from runtime system performance. DCPI captures changes in resource usage over time, under the hypothesis that poisoned data affects model behavior during training.

(i) Performance Data Collection: In order to characterize the behavior of the device during training, we collect time-series performance metrics (denoted as Pf_k for device k) using system monitoring tools. These metrics include CPU utilization, memory usage, and other runtime indicators known to correlate with the structure and input complexity of ML models [93]. Samples are recorded at regular intervals and collected without requiring access to application-level internals. Importantly, measurements are taken when the training task is prioritized on the device, reducing noise from background processes.

(ii) Data Modeling and Change Point Detection: In order to analyze fluctuations in the performance time series, we apply the Pruned Exact Linear Time (PELT) algorithm [36], an efficient method for detecting change points in sequential data. PELT minimizes a cost function that balances fit quality and model complexity. We use an ℓ_2-based cost function (costL2) to detect shifts in both the mean and variance of the signal. Based on the detected change points, we compute the Change Point Index (CPI) for each device as the normalized frequency of changes over a training duration $\mathcal{T}$:

$$CPI(X(t)) = \frac{1}{\mathcal{T}} \sum_{t=1}^{\mathcal{T}} CP(X(t)) \tag{1}$$

The DCPI threshold ($DCPI_{thresh}$) is defined as the absolute difference between CPI values in two reference states: $CPI(X(t))_{\text{idle}}$: collected when the device is idle, $CPI(X(t))_{\text{clean}}$: collected during training on clean (non-poisoned) data

$$DCPI_{thresh} = |CPI(X(t))_{\text{idle}} - CPI(X(t))_{\text{clean}}| \tag{2}$$

The device-specific DCPI is then computed by comparing the observed CPI during training to this threshold:

$$DCPI_k = |CPI(X(t))_k - DCPI_{thresh}| \tag{3}$$

A high $DCPI_k$ value suggests a significant behavioral deviation from the clean operation, potentially due to poisoned training inputs. The estimation of the $DCPI_{thresh}$ is crucial, especially when the device may be engaged in other processing activities that can influence the performance metrics. In such scenarios, the absolute difference between $CPI(X(t))_{idle}$ and $CPI(X(t))_{clean}$ nullifies the impact of third-party activity, such as other software processes. $DCPI_k$ serves as the primary metric derived from the

performance metrics on a specific device, Pf_k, to detect poisoning attacks. Algorithm 1 summarizes the overall workflow for detecting data poisoning attacks. In the algorithm, the central server aggregates the global weights, and the $DCPI_k$ calculated on each client is evaluated to identify performance abnormalities concerning normal device state execution.

Algorithm 1: DMLDetect($ServerExecute$)

INPUT: Clients: k, Fraction of Clients per round: C
OUTPUT: Global weight w_{t+1}

for each round $t = 1$ to ∞ **do**
  $m \leftarrow max(C * K, 1)$
  $S_t \leftarrow$ set of m clients
  for each client $k \in S_t$ in parallel **do**
    $W_{t+1}^k \leftarrow ClientUpdate(k, w_t)$
    Detect CP_k in Pf_k of k
    Calculate the $CPI(X(t))_k$ of as in the Equation 1
    Calculate the $DCPI_{thresh}$ as Equation 2
    Calculate DCPI of k, $DCPI_k$ as in the Equation 3
    if $DCPI_k > DCPI_{thresh}$ **then**
      Calculate the amount of Poisoning in k using ML
    end if
  end for
  $w_{t+1} \leftarrow \sum_{k=1}^{\mathcal{K}} \frac{n_k}{n} w_{t+1}^k$
end for

(iii) Identification and Classification: In order to evaluate whether runtime metrics can help identify poisoned devices, we build a binary classification dataset. Identification and Classification. Each entry in this dataset represents a single client device. The input features include runtime statistics such as CPI, DCPI, and the global model's accuracy and loss during training. The output label indicates whether the device was poisoned or clean, based on the setup used in the following experiments. Two classical ML models are used for classification: Random Forest and k-Nearest Neighbors. These models are trained separately for FL and SL tasks. This classification pipeline is evaluated in later experiments, where we apply DCPI under a variety of poisoning scenarios and measure its effectiveness as a lightweight detection mechanism in decentralized training.

5.5 Implementation of FL and SL Using the Flower Framework

In order to implement and coordinate both FL and SL, in our physical testbed experiments, we select the Flower framework[5]. Flower is a flexible, open-source platform designed for

[5] `https://flower.ai/`

FL research. It supports a range of learning paradigms, including FL and SL, and provides seamless integration with standard ML libraries such as PyTorch and TensorFlow. Flower provides high-level abstractions that simplify the development of client-server interactions, model aggregation strategies, and training orchestration. Its ability to custom training loops makes it suitable for our resource-constrained edge environment built with Raspberry PI devices. For FL, we use Flower's built-in $NumPyClient$ class, allowing each Raspberry PI to act as a local training node. Figure 13 shows the learning flow. The client's $fit()$ function trains a local model on its dataset using received global weights and returns the updated weights to the server. The server then uses the $aggregate\_fit()$ function to combine these updates and produce a new global model. Here, we use the FedAvg algorithm, which is selected from Flower's configurable set of strategies. For evaluation, clients run $evaluate()$ on their local test set and return performance metrics, which the server aggregates using $aggregate\_evaluate()$. We adopt this standard coordination flow with only minimal modification.

Unlike FL, SL requires a more customized implementation that overrides the default functions. On the client side, we redefined the $fit()$ function to include only the forward pass through the client-side model layers, generating intermediate embeddings from the local training set. These embeddings were sent to the server. On the server side, we overrode the $aggregate\_fit()$ method to complete the remaining forward pass, compute the training loss, and backpropagate gradients with respect to the received embeddings. These gradients were returned to the corresponding clients. On the client side, the $evaluate()$ function was repurposed to perform the backward pass using these gradients. After updating the client-side model, the client also ran a validation forward pass using its local validation set. The resulting embeddings were returned to the server for final evaluation. Finally, on the server side, we customized $aggregate\_evaluate()$ to perform the remaining forward pass on the validation embeddings, compute performance metrics (e.g., accuracy, loss), and record the aggregated result. This reinterpreted training and evaluation flow allowed us to implement SL within the same client-server communication model as FL, without modifying Flower's core infrastructure. The adapted process is visualized on the right in Figure 13.

5.6 Experiment 1: Distributed ML under Clean and Poisoned Conditions

This experiment aims to understand how different types of data poisoning affect DML in edge environments. Using both FL and SL settings, we analyze how poisoning influences the global model's performance and collect the runtime behavior of each client under such conditions. This experiment serves as a foundation for subsequent analysis of abnormal behavior detection. The experiments were conducted on a physical testbed composed of 10 Raspberry Pi 4 Model B devices acting as clients, and a laptop server running

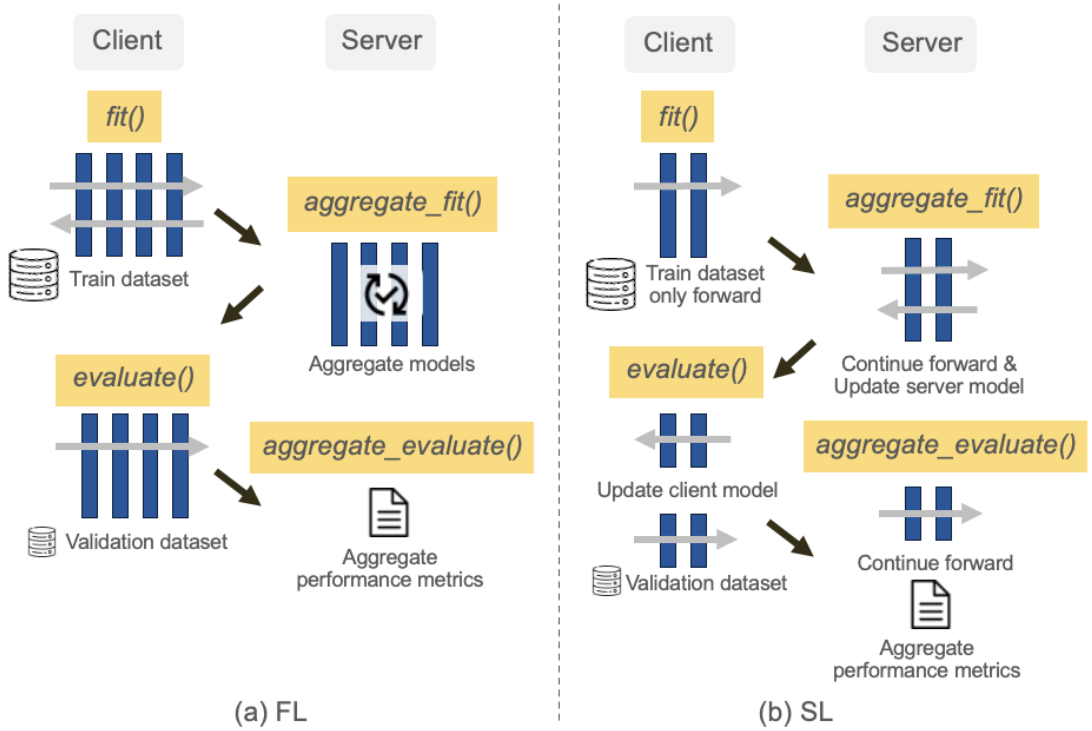


Figure 13. Default FL flow in the Flower framework and the SL flow we implement. We reinterpreted the default functions to exchange embeddings and gradients instead of full model parameters, enabling SL within the same communication structure.

Ubuntu 22.04 with a 4-core CPU coordinating the training. The devices were connected using LAN cables to ensure low latency and stable communication, with an average round-trip time (RTT) of approximately 102.5 milliseconds. All systems operated in a temperature-controlled environment (approximately 23°C) regulated by a thermostat. This environmental control was important because the CPU temperature fluctuations serve as one of the primary signals. By reducing external thermal noise, we accurately catch temperature readings changing as training-related behavior rather than ambient variation. The training process was orchestrated using the Flower framework. In order to collect run-time performance metrics, we used Telegraf[6], a lightweight monitoring agent capable of tracking system-level statistics. Metrics such as CPU usage, memory consumption, disk I/O, and CPU temperature were recorded during training and stored in InfluxDB as time-series data for further analysis.

As a representative real-world scenario inspired by the prompt *"Can we create a system to recognize unfamiliar objects or signs around town using local footage?"*, we simulated a collaborative learning task among autonomous drones performing object clas-

[6]https://www.influxdata.com/time-series-platform/telegraf/

sification in cluttered environments, such as urban roads or indoor warehouses. For this task, we used the TrashNet dataset [94], which contains labeled images of six categories of waste: cardboard, glass, metal, paper, plastic, and trash. This dataset simulates noisy, unconstrained visual input conditions and is commonly used to benchmark classification robustness in resource-limited perception tasks. In order to emulate a distributed training setup, the dataset was first replicated to each client device and then partitioned locally into 10 equal subsets using a shared deterministic seed. One partition was randomly selected and retained by each client to simulate local data ownership. While real-world FL and SL scenarios typically involve non-IID data, where each device collects unique data reflecting its specific environment, we adopted IID partitioning in this experiment to ensure controlled comparison across different poisoning conditions. The model used for training was a convolutional neural network (CNN) composed of three convolutional layers (with 32, 64, and 32 filters, all with a kernel size of 3), followed by three fully connected layers (with 64, 32, and 6 output neurons, respectively). The final layer uses six neurons to match the six output categories in the TrashNet dataset. Each convolutional layer was followed by maximum pooling, and a dropout rate of 20% was applied in the fully connected layers to mitigate overfitting. The model consisted of approximately 2 million trainable parameters, with an estimated computational cost of 200 million FLOPs.

Attack simulation: In order to simulate poisoning attacks, we implemented the four attack types introduced in the previous section: *blurring*, *occlusion*, *steganography*, and *label flipping*. Poisoning transformations were applied to each scientist's local samples before the training began, meaning the distributed learning process (FL or SL) operated on static, preprocessed datasets. By performing poisoning locally on each client, this approach reflects realistic scenarios where data corruption occurs during local collection on edge devices. The implementation of the poisoning attacks was as follows:

(i) Blurring: Gaussian blur was introduced using *PIL.ImageFilter.GaussianBlur()* with a fixed radius of 2, simulating low-quality image capture

(ii) Label flipping: A specified percentage of class labels was randomly reassigned to incorrect classes, ensuring the new label differed from the original.

(iii) Occlusion: Visual obstruction was simulated using *torchvision.transforms.RandomErasing*. The transformation was applied with a probability of 1.0, a scale range of (0.02, 0.33), an aspect ratio range of (0.3, 3.3), and a constant fill value of (250, 250, 250), representing a bright rectangular patch.

(iv) Steganography: A short secret message was embedded into image pixel values using the Least Significant Bit (LSB) encoding. Binary data were written in the LSBs of the red, green, and blue channels. A delimiter string ('#####') was appended to signal the end of the message. Each modification was verified by decoding the image to ensure full message recovery.

The intensity of poisoning was controlled by two parameters: (1) the poisoning ratio,

or percentage of a client's dataset that was poisoned, and (2) the number of poisoned clients. The poisoning ratios were set to 5%, 10%, 15%, 20%, 25%, and 30%. For each ratio, we also varied the number of poisoned clients across 1, 4, 7, and 10 devices to simulate both sparse and widespread poisoning. In the FL configuration, each client trained its local model for 5 epochs per round, using FedAvg for aggregation. Global model updates were computed by weighted averaging across participating clients. Early stopping was used with a patience of 5 rounds to avoid overfitting and unnecessary computation. In the split learning setup, the model was divided between clients and the server: clients hosted the feature extractor (3 convolutional layers), while the server handled the classifier (fully connected layers). SL rounds were configured with 1 epoch per client per round, and early stopping was applied with a patience of 10 rounds. «ADD TRIALS ,FL 3 TRIAL, SL clean 3, POISON 1.»

5.7 Experiment 2: Poisoning Detection via DCPI and Runtime Classification

This experiment evaluates the effectiveness of DCPI as a behavioral indicator of data poisoning in DML systems. Building on the run-time performance data collected in Experiment 1, we apply the DCPI framework introduced in Section 5.4 to both FL and SL settings. Across these configurations, we assess whether poisoned clients can be reliably detected through detailed analysis of DCPI values.

DCPI computation: In order to compute DCPI, we use the PELT algorithm to identify change points in device behavior. For FL, where training rounds are longer, the PELT algorithm was configured with a window size of 2 and a skip value of 5. For SL, which has shorter rounds, a window size of 1 and a skip value of 1 were used. Following change point detection, we computed the CPI and the corresponding DCPI for each client device. The DCPI threshold ($DCPI_{thresh}$) was calculated based on two baseline conditions. First, the CPI was measured during an idle phase, in which the devices were left powered on but did not perform any training for a one-hour interval. Second, CPI was measured during clean training, using non-poisoned data for 10 rounds. The absolute difference between these two values formed the threshold for detecting abnormal fluctuations. Any device whose observed training CPI significantly deviated from this threshold was flagged as exhibiting potentially poisoned behavior.

Training the classifier: In order to evaluate whether run-time behavior could reliably indicate poisoning, we constructed a classification dataset in which each training round from each device was treated as a sample. The dataset included features: the device's DCPI, CPI, global model loss, and validation accuracy for the round. Global training metrics are included because they are typically accessible at the server level without requiring access to raw client data. Each sample was labeled as either poisoned or clean, depending on the poisoning configuration used during that training run. We trained

two lightweight classification models on this dataset: Random Forest and k-Nearest Neighbors, both implemented using the scikit-learn library. The models were trained using an 80:20 train-test split and evaluated separately for FL and SL scenarios. The aim was not to maximize predictive accuracy, but to assess whether behavioral features alone can support reliable poisoning detection under practical edge conditions.

Evaluation setup: The classifiers were trained and evaluated separately for FL and SL experiments. Each poisoning type (*blurring*, *occlusion*, *steganography*, *label flipping*) was tested independently across multiple poisoning ratios (5% to 30%) and client counts (1, 4, 7, 10). This ensured that the results were generalized across a range of poisoning intensities and distribution patterns. In total, 200 training sessions were logged and analyzed for the classification task. We then report the F1 score for each classifier.

5.8 Experiment 3: Generalizability and Robustness Evaluation

This final set of experiments evaluates the broader applicability and reliability of the DCPI-based detection method under various training conditions. Specifically, we examine whether it remains robust in the presence of benign background load on client devices and whether DCPI generalizes to different datasets, models, and learning setups.

Robustness to background load: In order to evaluate whether the proposed poisoning detection method remains reliable under realistic operating conditions, we conduct an additional experiment involving controlled background activity on the client devices. In practical edge deployments, devices are often tasked with multiple responsibilities, such as logging, telemetry, or system monitoring, that can introduce noise into performance metrics. This experiment tests whether such benign fluctuations impact the reliability of DCPI or alter the thresholding parameters used for detection. We reuse the same experimental pipeline as in Experiment 2, using the TrashNet dataset and the FL setup. However, this time we introduced artificial background load during training using the *stress-ng*[7], which is a widely used Linux benchmarking tool that can generate synthetic CPU, memory, and I/O load across multiple threads. We selected this tool for its fine-grained control over load intensity, type, and process count, allowing us to simulate realistic multitasking conditions on edge devices. We vary the load intensity by launching 4, 8, and 12 concurrent background processes on selected clients during training. Each background process was configured to consume about 10% of a CPU core. The Raspberry Pi 4, with its quad-core CPU, already runs over 200 lightweight background tasks under typical conditions. In our baseline FL setup, training alone used roughly 70% of the total CPU capacity. To simulate full CPU saturation in a realistic multitasking scenario, we used *stress-ng* to launch 12 additional processes, gradually filling the remaining 30% headroom. Together, the training workload and these simulated tasks brought the system near full utilization. In order to maintain consistency, the only difference from

[7] `https://github.com/ColinIanKing/stress-ng`

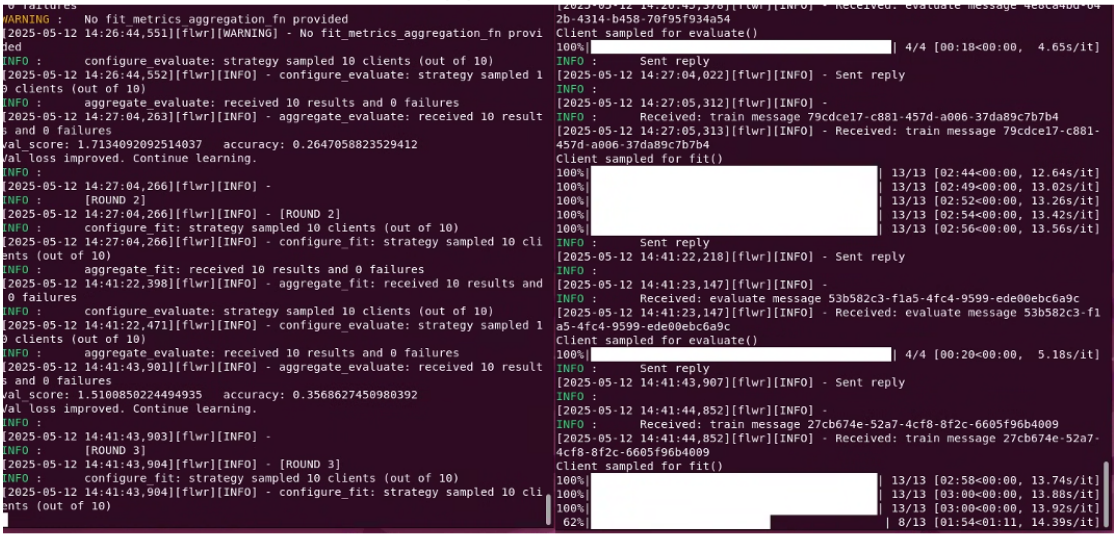
the previous experiment is the introduction of background processes on selected clients. As before, the 4 poisoning types were tested individually across poisoning ratios of 5%, 15%, and 30%, while performance metrics were collected throughout training. DCPI was computed following the same procedure, allowing for direct comparison. Importantly, we examined shifts in the CPI_{idle} and CPI_{thresh} baselines under load to check whether the presence of background load affected the reliability of our detection metric.

Generalizability to other datasets: In order to assess the transferability of the detection method, we repeated the poisoning detection pipeline using the Chinese Traffic Sign Recognition Benchmark [8] as an additional scenario. This dataset represents a different category of visual input, focusing on structured signs in public environments rather than unstructured waste. It simulates a use case where autonomous drones collaboratively train a perception model to navigate urban roadways and intersections. The distinct visual structure and application context make it suitable for assessing the robustness of our detection pipeline beyond the original TrashNet setting. Moreover, this experiment also introduced a shift in learning style. Unlike the previous experiments that used a fully trained CNN from scratch, here we employed a pre-trained MobileNetV3-Small model[9], split after its final bottleneck block, and fine-tuned it in an SL configuration. The decision to use a pre-trained model reflects a common real-world setting, where edge devices fine-tune a base model on locally collected data. On the client side, the model produced feature maps of size 7×7×96, which were then forwarded to the server for downstream classification processing with dimensions 7×7×576. The same four poisoning techniques: (*blurring*, *occlusion*, *steganography*, *label flipping*), were applied to simulate corrupted sensor input. DCPI was computed using the same monitoring framework and change point detection algorithm as in the previous experiment.

[8] https://nlpr.ia.ac.cn/pal/trafficdata/recognition.html

[9] https://huggingface.co/qualcomm/MobileNet-v3-Small

(a) Overall testbed



(b) Server Log

(c) Client Log

Figure 14. 10 Raspberry PIs used DML experiments; (a) overall setup; (b) server log; (c) client log

6 Results

6.1 Result 1: Distributed ML under Clean and Poisoned Conditions

We observed how data poisoning impacted model performance under both FL and SL paradigms. Poisoned data led to measurable drops in accuracy and increased convergence instability across all configurations. These findings establish a baseline understanding of how model behavior changes under adversarial input and set the stage for the anomaly detection evaluation presented in Result 2.

Baseline accuracy without poisoning: To establish a baseline, the CNN model was first trained under clean conditions. The FL configuration achieved 73.6% test accuracy, which aligns closely with centralized training benchmarks reported in previous TrashNet studies. In the SL setup, where the model is split between clients and a central server, test accuracy reached 53.7%. Performance degradation in SL can be attributed to suboptimal partitioning and communication delays across multiple clients. Notably, when SL was executed with a single client, accuracy improved to 71.8%, highlighting the impact of distributed coordination in SL. Although SL performance was lower in multi-client settings, the model was still accurate enough to support our main goal of analyzing behavioral differences introduced by poisoning.

Impact of poisoning on model accuracy and training stability: Table 2 summarizes the impact of increasing poisoning ratios and varying numbers of poisoned clients. As the proportion of poisoned data rose from 5% to 30%, model accuracy declined steadily. Similarly, the performance degraded more severely as the number of poisoned clients increased from one to ten. Both trends were statistically significant. A Friedman test confirmed the effect of poisoning rates on accuracy in FL ($\chi^2(2) = 20.0$, $p < .05$, $W = 0.22$) and SL ($\chi^2(2) = 9.175$, $p < .05$, $W = 0.20$). The number of poisoned clients also significantly impacted model accuracy (FL: $\chi^2(2) = 70.9$, $p < .05$, $W = 0.66$; SL: $\chi^2(2) = 28.9$, $p < .05$, $W = 0.48$). Training convergence was also affected. In clean settings, both FL and SL typically converged within 40 rounds. However, under high poisoning conditions (e.g., 10 poisoned clients at 30% poisoning), FL showed early stopping around round 7 (with a patience of 5), while SL required up to 20 rounds (with a patience of 10) to stabilize. Figure 15 illustrates these convergence trends, highlighting how poisoning disrupts learning dynamics in FL and SL.

Poisoning disrupts gradient dynamics: We compared the gradients from clean and poisoned training by calculating the root mean square error (RMSE) across different layers to understand how poisoned data affects learning. The CNN model consisted of three convolutional layers (named *conv1*, *conv2*, and *conv3* with 32, 64, and 32 filters, respectively) and three fully connected layers (*fc1*, *fc2*, and *fc3* with 64, 32, and 6 output units). As poisoning levels increased, RMSE values rose sharply, indicating more unstable gradient flows. Figure 16 shows this trend across convolutional and fully

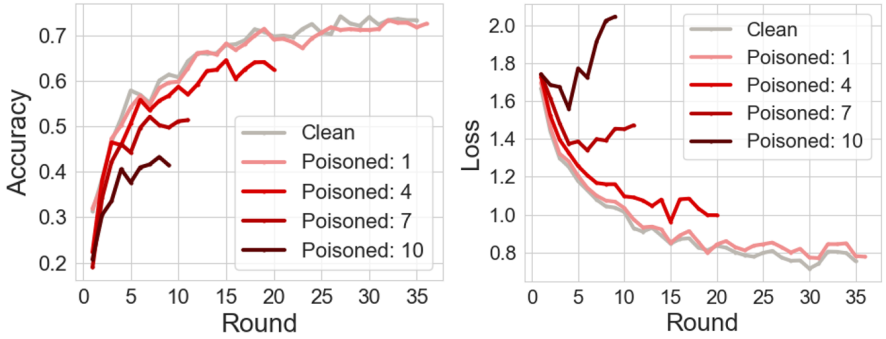


Figure 15. Learning curves under a blurring setting with 30% poisoning rate show poisoning affects model convergence.

Table 2. Model performance degradation as incremental data poisoning is introduced by individual devices gradually.

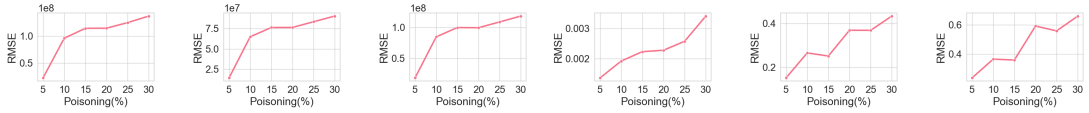
(a) FL

Poisoning(%)	The number of devices			
	1/10	4/10	7/10	10/10
un-poisoned	73.6 ± 0.5	73.6 ± 0.5	73.6 ± 0.5	73.6 ± 0.5
Poisoning Type	**Blurring**			
5	72.7 ± 1.9	63.7 ± 0.7	53.7 ± 0.7	46.3 ± 1.1
10	69.4 ± 1.0	65.0 ± 2.7	51.9 ± 1.1	46.3 ± 1.6
15	72.5 ± 1.9	68.1 ± 1.8	53.7 ± 0.7	43.3 ± 1.0
20	70.8 ± 1.6	65.2 ± 1.1	53.2 ± 0.5	46.4 ± 1.3
25	70.7 ± 1.2	62.9 ± 0.5	52.4 ± 1.7	47.2 ± 1.1
30	70.5 ± 1.8	62.5 ± 1.0	51.4 ± 0.3	41.4 ± 0.4
Poisoning Type	**Occlusion**			
5	70.5 ± 1.0	66.9 ± 0.9	60.1 ± 2.9	49.4 ± 1.7
10	71.0 ± 0.4	67.2 ± 1.9	54.2 ± 0.6	46.6 ± 1.0
15	71.2 ± 1.5	66.8 ± 2.0	60.3 ± 0.3	46.4 ± 1.1
20	69.6 ± 0.6	67.2 ± 1.9	56.5 ± 0.9	48.1 ± 0.4
25	71.6 ± 1.1	66.9 ± 3.0	55.0 ± 0.5	44.0 ± 0.4
30	70.1 ± 0.3	66.8 ± 0.3	51.0 ± 0.0	46.1 ± 1.8
Poisoning Type	**Steganography**			
5	71.5 ± 0.7	64.4 ± 0.6	55.9 ± 1.0	51.5 ± 0.7
10	72.9 ± 0.7	64.8 ± 0.2	56.3 ± 0.4	50.5 ± 1.0
15	71.9 ± 0.4	66.5 ± 0.4	53.7 ± 0.7	49.2 ± 1.7
20	72.0 ± 0.6	66.7 ± 1.1	54.5 ± 0.4	49.6 ± 1.7
25	72.1 ± 0.9	65.4 ± 0.6	54.3 ± 0.5	50.7 ± 0.6
30	70.8 ± 0.8	65.6 ± 0.3	53.9 ± 0.4	51.2 ± 1.0
Poisoning Type	**Label Flipping**			
5	70.2 ± 1.0	62.9 ± 1.5	53.2 ± 0.5	50.3 ± 0.9
10	71.9 ± 1.1	64.2 ± 1.0	55.7 ± 1.6	47.6 ± 0.4
15	72.2 ± 0.4	63.9 ± 1.2	50.7 ± 0.5	45.9 ± 0.4
20	71.2 ± 1.7	63.2 ± 1.3	47.7 ± 1.7	40.3 ± 0.5
25	71.0 ± 1.1	61.9 ± 0.6	45.4 ± 1.3	38.8 ± 1.2
30	70.1 ± 1.2	59.3 ± 0.5	43.1 ± 0.5	36.6 ± 0.3

(b) SL

Poisoning(%)	The number of devices			
	1/10	4/10	7/10	10/10
un-poisoned	53.7 ± 1.5	53.7 ± 1.5	53.7 ± 1.5	53.7 ± 1.5
Poisoning Type	**Blurring**			
5	48.2	45.9	43.9	35.6
10	48.4	41.0	33.3	33.9
15	50.4	38.6	35.9	37.5
20	49.6	42.4	38.2	32.5
25	51.0	31.6	38.8	34.9
30	46.9	43.3	38.8	34.7
Poisoning Type	**Occlusion**			
5	50.6	46.1	42.5	39.2
10	50.4	43.3	42.0	39.0
15	49.6	46.7	41.4	38.8
20	49.2	41.0	38.0	33.7
25	48.0	41.4	37.8	31.4
30	49.0	38.8	35.1	31.8
Poisoning Type	**Steganography**			
5	56.9	52.5	46.7	41.6
10	52.5	47.6	40.4	41.4
15	53.1	45.3	41.2	42.0
20	44.9	44.3	38.2	41.0
25	49.8	48.8	37.1	44.1
30	52.2	46.1	45.7	41.2
Poisoning Type	**Label Flipping**			
5	52.4	43.1	49.8	45.7
10	48.6	45.5	46.9	43.3
15	55.5	47.1	48.8	44.1
20	45.7	46.3	46.9	38.6
25	48.4	42.7	38.8	41.8
30	53.4	48.0	49.8	43.7

connected layers. Pearson correlation coefficients confirmed the relationship between poisoning severity and gradient distortion: *conv1* ($r = 0.86$), *conv2* ($r = 0.85$), *conv3* ($r = 0.85$), *fc1* ($r = 0.96$), *fc2* ($r = 0.96$), and *fc3* ($r = 0.95$), all with $p < .05$. These results suggest that poisoned inputs trigger noisy or conflicting gradient updates, which hinder convergence and degrade generalization.



(a) *conv1* (b) *conv2* (c) *conv3* (d) *fc1* (e) *fc2* (f) *fc3*

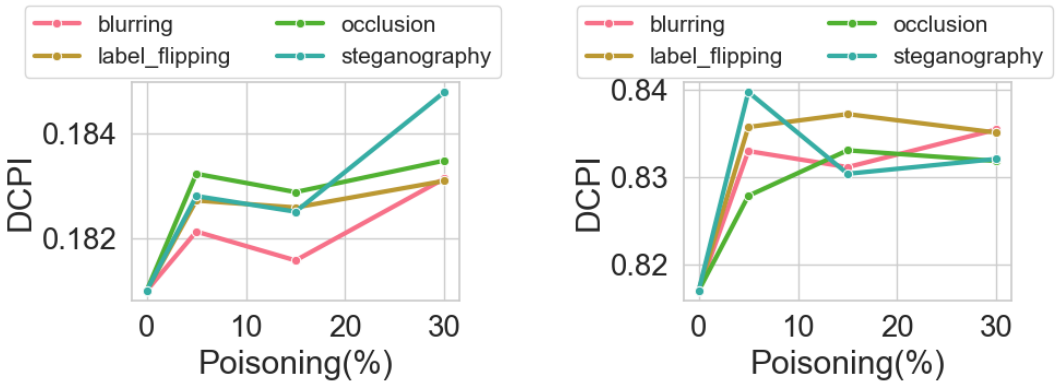
Figure 16. RSME of the gradient changes of the model during training with different levels of poisonings

6.2 Result 2: Poisoning Detection via DCPI and Classification

This section evaluates the effectiveness of the DCPI metric in identifying poisoned clients. By applying DCPI across multiple attack types and poisoning levels, we assess its sensitivity to behavioral anomalies and examine whether it reliably distinguishes between clean and compromised devices. Our analysis also explores device-level DCPI values.

DCPI behavior in FL and SL: Figure 17(a) shows how DCPI responds to different poisoning methods under the FL setting. For each poisoning configuration, DCPI was calculated separately for poisoned and clean clients within the same training session. For example, if clients 1 to 7 were poisoned, their DCPI values are recorded for that specification, while the remaining clean clients (clients 8 to 10) served as 0% reference. This setup allowed for consistent comparisons between poisoned and clean behavior in identical environments. Also, as described in Equation (1) in Section 5.4, DCPI is calculated by dividing the number of detected change points by the training duration T. For fair DCPI measurement across all trials, we cropped the monitoring window to align strictly with each client's local training phase, excluding server aggregation time. As shown in Figure 17(a), DCPI consistently increased as the poisoning severity rose from 5% to 30% in the FL condition. The strongest trend was observed for *steganographic* poisoning, followed by *blurring* and *occlusion*. Pearson correlation analysis confirmed the positive relationship between poisoning level and DCPI: $r = 0.92$ for steganography, $r = 0.84$ for both blurring and occlusion, and $r = 0.76$ for label flipping. A Friedman test verified the statistical significance of DCPI differences across poisoning levels ($p < 0.05$), validating DCPI's effectiveness in FL deployments. Figure 17(b) presents the same DCPI analysis applied to SL. As with FL, only poisoned devices were used for each poisoning specification, and clean devices within the same configuration provided the 0% reference. We also cropped the monitoring window to align with each client's local training phase. A Mann-Whitney U test revealed a statistically significant difference between clean and poisoned devices ($U = 0.0$, $p < 0.05$), reinforcing DCPI's generalizability across distributed learning paradigms.

Device-level DCPI characterization: We also examined device-level DCPI behavior by analyzing a trial in which 7 out of 10 devices were poisoned with 30% blurring. Figure 18 shows the average DCPI per device, computed over 20 training rounds. In this particular trial, the average DCPI values for poisoned devices did not exceed those of



(a) FL (b) SL

Figure 17. DCPI variations in FL and SL under different poisoning types and levels.

clean devices. However, the poisoned devices exhibited higher variability in their DCPI values, which his consistent with the expectation that poisoning introduces instability in runtime behavior. This pattern was confirmed by variance analysis, which showed higher dispersion among poisoned devices $4.55e-05$ compared to clean ones $3.97e-05$. A Mann-Whitney U test confirmed a statistically significant difference between the two groups ($U = 12532$, $p < 0.05$), and Wilcoxon-Bonferroni tests found no significant differences among clean devices ($p < 0.05$), indicating stable behavior under normal conditions. These findings suggest behavioral monitoring is a reliable indicator of anomalous nodes.

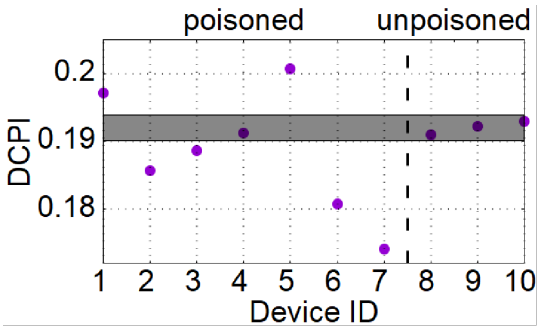


Figure 18. Device-wise DCPI values under 30% blurring attack.

Classification performance using runtime metrics: We constructed separate binary classification datasets for FL and SL using DCPI, CPI, and global training metrics (loss and accuracy) as input features. Each data sample corresponded to a single client training round and was labeled as either "poisoned" or "clean," based on the experimental configuration described in Section 5.6. In the FL configuration, we collected approximately

Table 3. <Binary Case "Poisoned or Not-poisoned", TrashNet> Classification accuracy (%) for predicting data poisoning attacks (P); Random Forest (RF) and K-nearest Neighbour (KNN).

Test (Model data → Predicted)	RF		KNN		Average	
	FL	SL	FL	SL	FL	SL
(DCPI) → P	0.77	0.81	0.75	0.78	0.76	0.80
(DCPI,Accuracy) → P	0.77	0.72	0.70	0.66	0.74	0.69
(DCPI,CPI) → P	0.77	0.81	0.75	0.77	0.76	0.79
(DCPI,CPI,Accuracy) → P	0.83	0.76	0.76	0.71	0.80	0.74
(DCPI,CPI,Accuracy, Loss) → P	0.87	0.78	0.73	0.65	0.80	0.72

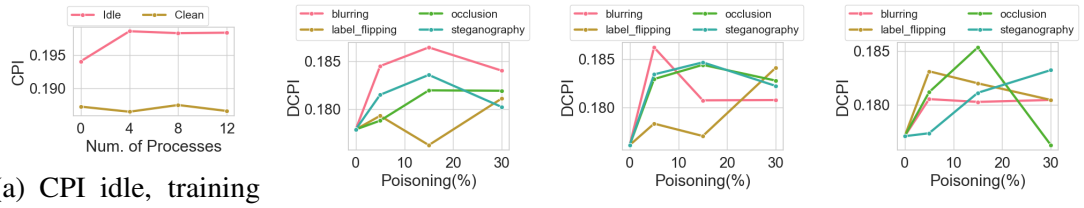
120,000 labeled samples across multiple trials, while the SL configuration included around 40,000 samples. For each setup, we trained two classical machine learning models, Random Forest and k-Nearest Neighbors, using an 80:20 train-test split. Table 3 summarizes the classification results. In FL, accuracy improved as more features were used, reaching up to 87% with the full feature set (DCPI, CPI, accuracy, and loss). In contrast, SL achieved strong performance even with DCPI alone, reaching 81% accuracy. Adding CPI or global metrics did not significantly improve performance. These results suggest that runtime metrics, particularly DCPI, can serve as effective features for detecting poisoned behavior during distributed training in both FL and SL contexts.

6.3 Result 3: Generalizability and Robustness Evaluation

This section presents two additional experiments designed to evaluate the robustness and generalizability of the proposed poisoning detection method. The first experiment, conducted under the FL setting, introduced controlled background processes to simulate realistic system noise and multitasking. The second experiment tested generalizability in the SL setting by applying the same detection pipeline to a different dataset and model architecture.

Robustness to background processes: In order to evaluate the system's resilience in realistic conditions, we applied controlled background workloads (4, 8, 12 processes) using *stress-ng* tool while running FL training. Figure 19(a) shows the CPI_{idle} and CPI_{clean} values measured during idle and clean FL training under different levels of background load. These values are used to compute $DCPI_{\text{thresh}}$. The figure illustrates that CPI remains stable even when moderate background processes are running. This stability is important because it means normal system activity does not distort the baseline. As a result, DCPI can reliably flag true poisoning anomalies without being misled by routine multitasking or load fluctuations. Figures 19(b–d) show DCPI variations under different poisoning methods and background load intensities. Despite the added stress, DCPI still effectively distinguishes poisoned from clean clients across all process levels. Statistical testing confirmed this: 4 processes ($U = 4.0$, $p < 0.05$), 8 processes ($U = 0.0$,

$p < 0.05$), and 12 processes ($U = 4.0$, $p < 0.05$). Interestingly, some inconsistency was observed in the label flipping attack. Compared to image-based attacks such as *blurring* and *steganography*, *label flipping* introduced less pronounced performance changes, likely because it perturbs gradient direction less than pixel-level modifications. Still, DCPI remained sensitive enough to detect poisoning reliably in most conditions.



(a) CPI idle, training (clean) (b) 4 process (c) 8 process (d) 12 process

Figure 19. CPI and DCPI with different numbers of background processes.

Generality across datasets and models: We repeated the poisoning detection experiment using a different image dataset, a different model architecture (MobileNetV3-Small), and a different training strategy (fine-tuning). As shown in Figure 20, DCPI values increased consistently with higher poisoning ratios. A Mann-Whitney U test confirmed a statistically significant difference between poisoned and clean conditions ($U = 0.0$, $p < 0.05$), consistent with earlier findings. This supports the generalizability of DCPI as a universal signal of data-induced training disruptions. Using the same classification pipeline as in previous experiments, we evaluated detection performance. Table 4 shows that DCPI alone achieved 80% accuracy on average, and combining it with CPI improved classification slightly to 81%. These results mirror the trends from TrashNet experiments, indicating that DCPI can extend effectively to new training tasks and visual domains.

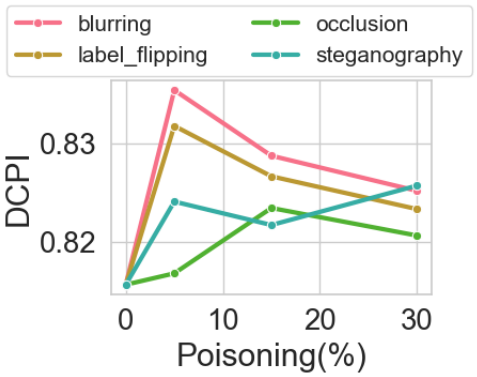


Figure 20. DCPI response to poisoning under SL with Chinese Traffic Sign dataset.

Table 4. <Binary Case "Poisoned or Not-poisoned", Chinese Dataset> Classification accuracy (%) for predicting data poisoning attacks (P); Random Forest (RF), Support Vector Machine (SVM) and K-nearest Neighbour (KNN).

Test (Model data → Predicted)	RF	SVM	KNN
(DCPI) → P	0.81	0.52	0.78
(DCPI,Accuracy) → P	0.69	0.69	0.67
(DCPI,CPI) → P	0.82	0.60	0.79
(DCPI,CPI,Accuracy) → P	0.78	0.69	0.74
(DCPI,CPI,Accuracy, Loss) → P	0.79	0.68	0.70

6.4 Summary

The results demonstrate that data poisoning introduces measurable disruptions to both model performance and training dynamics in FL and SL. Even moderate levels of poisoning degrade accuracy and delay convergence, highlighting the importance of early detection. The proposed runtime monitoring framework, centered around DCPI, successfully detects such anomalies using only lightweight system-level signals. DCPI proved sensitive to various poisoning types and robust under noisy, multitasking conditions, without requiring access to training data or model internals. Furthermore, its effectiveness is generalized across datasets, model architectures, and learning setups, including fine-tuned models in SL. These findings validate the feasibility of Phases III and IV in the proposed system workflow, supporting scalable, decentralized learning with real-time safeguards for training integrity in edge environments.

7 Discussion

Our experiments reveal important insights, challenges, and opportunities in achieving effective Edge Intelligence on Command. Although we successfully demonstrated the potential of LLM-based infrastructure assembly and poisoning detection, several practical considerations remain.

Edge deployment feasibility of LLMs: Recent advances indicate that deploying LLMs on edge devices is becoming increasingly viable. Traditionally reliant on cloud-based computation, LLMs can now be adapted for local environments through model distillation and compression. These compact variants, often referred to as Small Language Models (SLMs), support efficient, low-latency decision-making directly on resource-constrained devices [28]. This local execution model not only reduces reliance on remote infrastructure but also improves responsiveness and energy efficiency. Meanwhile, larger cloud-based models can still handle complex orchestration tasks across many devices when needed. It is evident that when orchestrating thousands of devices, the computational demands for LLM assembly can grow considerably. Compact models represent a critical step in this direction, offering the potential for sustainable, scalable deployment of intelligence capabilities at the edge. The portability of modern edge platforms further supports flexible LLM deployment. For example, Microsoft's AI-powered backpack demonstrates how LLMs can be embedded into lightweight, mobile systems to enable on-the-go intelligence in previously inaccessible scenarios [45].

Context-aware resource management: Beyond static hardware specifications, LLMs offer latent reasoning capabilities that can support context-aware infrastructure decisions. In real-world environments, optimal device selection often depends on situational factors such as location dynamics, time of day, or user mobility. For example, devices in crowded public spaces may be less reliable due to shorter dwell times or fluctuating availability. To handle such scenarios, LLMs can be augmented with real-time contextual data. For example, environmental conditions, user presence, or mobility patterns enable LLMs to reason not only about device capability but also availability and stability. This is especially valuable in opportunistic settings like transit hubs or public IoT deployments, where connections are short-lived. Integrating lightweight environmental signals or user interactions can improve the system's responsiveness to real-world conditions [37], allowing for more adaptive and resilient orchestration.

Operational trade-offs between open-source and proprietary LLMs: While LLMs enable flexible orchestration, their use in infrastructure assembly raises practical trade-offs between open-source and proprietary models. Open-source LLMs offer greater customization and no licensing costs, but require ongoing maintenance as new models emerge. Keeping them updated often demands manual intervention or fine-tuning, which can limit scalability. Techniques like incremental learning may help, but their cost-effectiveness remains uncertain. In contrast, proprietary models offload infrastructure

management to the provider, simplifying deployment and maintenance. However, they introduce usage fees and vendor lock-in risks, where changes in pricing, policy, or service availability may reduce autonomy or disrupt operations. Choosing between these options involves balancing flexibility, control, and long-term operational cost.

Lack of explicit optimization in LLMs: While LLMs can produce context-aware decisions through pattern recognition and reasoning, they lack explicit optimization capabilities. Traditional techniques, such as heuristics, linear programming, or reinforcement learning, are designed to optimize resource allocation under well-defined constraints, offering predictable and repeatable outcomes [30]. In contrast, LLM-based decisions are inherently probabilistic and less reliable in resource-constrained or latency-critical scenarios. Bridging this gap may require hybrid systems that combine LLMs with structured optimization frameworks or refined prompting strategies to encourage more consistent and resource-aware decision-making. However, LLMs provide complementary value in domains where optimization targets are difficult to define or quantify. For example, as seen in Section 3.2.1, actual imaging quality is not always reflected by device specifications like resolution. Instead, it depends on complex interactions between hardware (e.g., sensor size, lens type) and software (e.g., image processing pipelines), which are hard to formalize into standard benchmarks. In such cases, LLMs can leverage broad contextual knowledge to make informed judgments. This makes LLMs especially useful in opportunistic edge scenarios where decisions must account for loosely defined or context-specific constraints that conventional optimizers cannot easily capture.

Trust and reliability of LLMs: While LLMs show promise for managing decentralized infrastructures, their opaque internal reasoning poses serious concerns for reliability, transparency, and accountability. These models are inherently black-boxes, making them unsuitable for critical applications without additional safeguards. To address this, emerging AI regulations such as the EU's General Data Protection Regulation (GDPR) and the AI Act emphasize principles like transparency, fairness, and human oversight [92]. Mechanisms such as Human-in-the-Loop (HITL), Human-on-the-Loop (HOTL), and Human-in-Command (HIC) define required levels of human intervention based on risk. Similar principles have been adopted globally through national strategies and standards from organizations like UNESCO and ISO. Decentralized systems built on LLMs must go beyond privacy protection to ensure explainability, robustness, and user control. This includes transparency-by-design and user interfaces that enable meaningful intervention. For example, one approach involves deploying public servers that perform computation without accessing raw user data, preserving ownership and autonomy. Additionally, any collaboration or model sharing must operate on a voluntary, opt-in basis with informed consent. Establishing trust in LLM-driven orchestration requires more than compliance. It demands systems that give power and oversight to users and align with evolving legal and ethical expectations.

Security and privacy challenges of LLMs: In parallel, the deployment of LLMs in

distributed systems brings forward substantial security and privacy challenges. As these models often process sensitive or context-specific data, they become high-value targets for malicious actors, including threats such as identity theft, malware injection, and poisoning attacks. These vulnerabilities are particularly concerning in decentralized environments where device coordination could be exploited to compromise entire networks. In order to mitigate these risks, secure computation mechanisms such as Trusted Execution Environments (TEEs) and Homomorphic Encryption (HE) are essential for preserving the integrity and confidentiality of infrastructure components[53]. Furthermore, embedding differential privacy techniques into LLM frameworks can help prevent unauthorized data leakage during training or inference. Advanced approaches to secure LLM-based systems also include optimizing inference with encrypted activations and leveraging distributed learning paradigms such as Federated Learning, Split Learning, and Reinforcement Learning. These methods can not only strengthen privacy guarantees but also reduce the risks associated with centralized model exposure [71].

Application scope of behavior monitoring: One of our proposed behavior monitoring systems' main advantages is its ability to operate. Without requiring access to model internals or centralized aggregation, it detects poisoning attacks in settings where privacy and communication constraints are critical. This makes it particularly well-suited for cooperative and opportunistic computing environments, such as autonomous drones, wearable devices, and IoT sensor networks. For example, its applicability extends to scenarios such as personalized learning frameworks, including Decentralized Federated Learning as a Service (DFLaaS), where device-level training and privacy are essential [35]. The system is designed to be hardware-agnostic, though its detection accuracy is strongest on single-purpose devices with stable task profiles. In such contexts, anomalous traces caused by data poisoning are more easily distinguishable from normal fluctuations. Moreover, the system aligns well with decentralized and privacy-preserving AI trends, as its on-device analysis avoids exposing sensitive data to external servers. This local-first approach can be further enhanced by integrating collaborative validation across multiple devices and leveraging secure computation frameworks, such as multi-party computation and Trusted Execution Environments (TEEs), to maintain both privacy and robustness during distributed analysis.

Limitations of behavior monitoring: The proposed behavior monitoring system faces several practical limitations that must be addressed to ensure robust performance across diverse deployment environments.. First, on multi-purpose devices like smartphones, benign background processes can introduce noise into performance metrics, mimicking the effects of poisoning and increasing the risk of false positives. Although the system is more accurate during low-activity periods, distinguishing signal from noise remains difficult. Bootstrapping with typical usage patterns or correlating behavior across devices may improve reliability [22]. Second, sharing performance metrics across devices or systems introduces privacy risks, even if raw data or model parameters are not

transmitted. Runtime signals such as CPU usage, memory load, or execution duration can unintentionally reveal information that can be indirectly inferred, such as the nature of the task, user activity patterns, or system vulnerabilities. In decentralized environments, where edge devices often operate in public or personal contexts, this leakage could be exploited for profiling or inference attacks. While local analysis helps mitigate this risk, broader integration into collaborative systems will require new strategies for secure, privacy-preserving metric sharing, such as TEEs. Lastly, hardware instability or failure can distort runtime signals and lead to misclassification. Pre-checks for device health can reduce this risk, but add operational complexity. These limitations emphasize the need for improved robustness, context-awareness, and secure data handling in future iterations of the system.

Toward full end-to-end integration: This thesis outlines a four-phase architecture for enabling LLM-driven orchestration and secure collaborative learning at the edge. While the system was evaluated primarily on Phases III and IV, focusing on runtime monitoring and anomaly detection, the earlier stages, including natural language task specification, device discovery, and distributed ML setup, were explored only as feasibility studies. In particular, although LLMs demonstrated the ability to recommend suitable devices and plan task-specific configurations, the actual deployment of executable code (e.g., initiating FL or SL across selected clients) remains unimplemented. Bridging this gap would require integrating a complete orchestration backend capable of translating user-level task descriptions into runnable configurations on heterogeneous devices. For example, frameworks such as AutoM3L [48] demonstrate how configuration files, device roles, and learning paradigms can be automatically generated and deployed, enabling seamless launch of training or inference jobs without manual intervention. Realizing this full pipeline will involve persistent service discovery, secure device onboarding, and robust interaction with real-time hardware interfaces. Achieving these capabilities is a critical direction for future work in making Edge Intelligence on Command fully operational and scalable in real-world edge environments.

8 Summary and Conclusion

This thesis introduced *Edge Intelligence on Command*, a unified framework designed to simplify the orchestration and security of DML across heterogeneous edge devices. By employing LLMs for interpreting natural language commands, the framework enables non-expert users to dynamically assemble decentralized infrastructures from opportunistically discovered devices without the need for predefined configurations or expert intervention. This capability was demonstrated experimentally through real-world hardware setups, validating the feasibility and effectiveness of LLM-driven device selection and collaborative task execution. In order to ensure the integrity of these opportunistic collaborative tasks, the framework further incorporates a lightweight, decentralized monitoring mechanism, which detects data poisoning attacks by analyzing fluctuations in device-level performance metrics such as CPU usage and temperature. Evaluations across federated and split learning scenarios and under various conditions showed that DCPI reliably identified anomalous behaviors in edge environments, thereby maintaining the robustness and reliability of distributed learning processes.

Collaboration has always been the engine of human progress. People thrive not by individual strength alone but through their ability to share knowledge, specialize in roles, and work toward common goals. The same principle applies to the digital world, where billions of devices already communicate within structured, function-driven systems. Yet, much of their potential remains unused; idle processing power, inactive sensors, and valuable data confined within individual devices. Edge collaboration can build on this opportunity by enabling devices to freely support computation, gather insights, and learn from local knowledge. By lowering the barrier to coordination and participation in distributed learning, it fosters flexible, autonomous collaboration that mirrors the dynamics of human society.

References

[1] Ashar Ahmad, Muhammad Saad, Mohammed Al Ghamdi, DaeHun Nyang, and David Mohaisen. Blocktrail: A service for secure and transparent blockchain-driven audit trails. *IEEE Systems Journal*, 16(1):1367–1378, 2021.

[2] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE communications surveys & tutorials*, 17(4):2347–2376, 2015.

[3] Badr AlKhamissi, Millicent Li, Asli Celikyilmaz, Mona Diab, and Marjan Ghazvininejad. A review on language models as knowledge bases. *arXiv preprint arXiv:2204.06031*, 2022.

[4] Mohammad S Ansari, Saeed H Alsamhi, Yuansong Qiao, Yuhang Ye, and Brian Lee. Security of distributed intelligence in edge computing: Threats and countermeasures. *The Cloud-to-Thing Continuum: Opportunities and Challenges in Cloud, Fog and Edge Computing*, pages 95–122, 2020.

[5] Evan Asfoura, Gamal Kassem, Belal Alhuthaifi, and Fozi Belhaj. Developing chatbot conversational systems & the future generation enterprise systems. *International Journal of Interactive Mobile Technologies*, 17(10), 2023.

[6] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. *Advances in neural information processing systems*, 30, 2017.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[8] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, 2006.

[9] Keyan Cao, Yefan Liu, Gongjie Meng, and Qimeng Sun. An overview on edge computing research. *IEEE access*, 8:85714–85728, 2020.

[10] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186, 1999.

[11] Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J Pappas, and Eric Wong. Jailbreaking black box large language models in twenty queries. *arXiv preprint arXiv:2310.08419*, 2023.

[12] Sheng Chen, Yang Liu, Xiang Gao, and Zhen Han. Mobilefacenets: Efficient cnns for accurate real-time face verification on mobile devices. In *Chinese conference on biometric recognition*, pages 428–438. Springer, 2018.

[13] Bram Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72. Berkeley, CA, USA, 2003.

[14] Marco Conti and Mohan Kumar. Opportunities in opportunistic computing. *Computer*, 43(01):42–50, 2010.

[15] Hongwei Cui, Yuyang Du, Qun Yang, Yulin Shao, and Soung Chang Liew. Llmind: Orchestrating ai and iot with llm for complex task execution. *IEEE Communications Magazine*, 2024.

[16] Shuiguang Deng, Hailiang Zhao, Weijia Fang, Jianwei Yin, Schahram Dustdar, and Albert Y Zomaya. Edge intelligence: The confluence of edge computing and artificial intelligence. *IEEE Internet of Things Journal*, 7(8):7457–7469, 2020.

[17] Josu Diaz-De-Arcaya, Juan López-De-Armentia, Raúl Miñón, Iker Lasa Ojanguren, and Ana I Torre-Bastida. Large language model operations (llmops): Definition, challenges, and lifecycle management. In *2024 9th International Conference on Smart and Sustainable Technologies (SpliTech)*, pages 1–4. IEEE, 2024.

[18] Qifei Dong, Xiangliang Chen, and Mahadev Satyanarayanan. Creating edge ai from cloud-based llms. In *Proceedings of the 25th International Workshop on Mobile Computing Systems and Applications*, pages 8–13, 2024.

[19] Kan Feng, Lijun Luo, Yongjun Xia, Bin Luo, Xingfeng He, Kaihong Li, Zhiyong Zha, Bo Xu, and Kai Peng. Optimizing microservice deployment in edge computing with large language models: Integrating retrieval augmented generation and chain of thought techniques. *Symmetry*, 16(11):1470, 2024.

[20] Huber Flores. Opportunistic multi-drone networks: Filling the spatiotemporal holes of collaborative and distributed applications. *IEEE Internet of Things Magazine*, 7(2):94–100, 2024.

[21] Huber Flores, Agustin Zuniga, Farbod Faghihi, Xin Li, Samuli Hemminki, Sasu Tarkoma, Pan Hui, and Petteri Nurmi. Cosine: Collaborator selector for cooperative multi-device sensing and computing. In *2020 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 1–10. IEEE, 2020.

[22] Huber Flores, Agustin Zuniga, Naser Hossein Motlagh, Mohan Liyanage, Monica Passananti, Sasu Tarkoma, Moustafa Youssef, and Petteri Nurmi. Penguin: aquatic plastic pollution sensing using auvs. In *Proceedings of the 6th ACM Workshop on Micro Aerial Vehicle Networks, Systems, and Applications*, pages 1–6, 2020.

[23] Asbjørn Følstad, Theo Araujo, Effie Lai-Chong Law, Petter Bae Brandtzaeg, Symeon Papadopoulos, Lea Reis, Marcos Baez, Guy Laban, Patrick McAllister, Carolin Ischen, et al. Future directions for chatbot research: an interdisciplinary research agenda. *Computing*, 103(12):2915–2942, 2021.

[24] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1322–1333, 2015.

[25] Othmane Friha, Mohamed Amine Ferrag, Burak Kantarci, Burak Cakmak, Arda Ozgun, and Nassira Ghoualmi-Zine. Llm-based edge intelligence: A comprehensive survey on architectures, applications, security and trustworthiness. *IEEE Open Journal of the Communications Society*, 2024.

[26] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2, 2023.

[27] Nakul Garg, Irtaza Shahid, Erin Avllazagaj, Jennie Hill, Jun Han, and Nirupam Roy. Thermware: Toward side-channel defense for tiny iot devices. In *Proceedings of the 24th International Workshop on Mobile Computing Systems and Applications*, pages 81–88, 2023.

[28] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[29] Ching-Nam Hang, Pei-Duo Yu, Roberto Morabito, and Chee-Wei Tan. Large language models meet next-generation networking technologies: A review. *Future Internet*, 16(10):365, 2024.

[30] Cheol-Ho Hong and Blesson Varghese. Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms. *ACM Computing Surveys (CSUR)*, 52(5):1–37, 2019.

[31] Chenyu Hou, Kai Jiang, Tiantian Li, Meng Zhou, and Jun Jiang. Co-active: an efficient selective relabeling model for resource constrained edge ai. *Wireless Networks*, pages 1–14, 2025.

[32] Andrew G Howard. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[33] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55, 2025.

[34] Beomyeol Jeon, SM Ferdous, Muntasir Raihan Rahman, and Anwar Walid. Privacy-preserving decentralized aggregation for federated learning. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6. IEEE, 2021.

[35] Kleomenis Katevas, Diego Perino, and Nicolas Kourtellis. Flaas-enabling practical federated learning on mobile environments. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, pages 605–606, 2022.

[36] Rebecca Killick, Paul Fearnhead, and Idris A Eckley. Optimal detection of changepoints with a linear computational cost. *Journal of the American Statistical Association*, 107(500):1590–1598, 2012.

[37] Sunyoung Kim, Jennifer Mankoff, and Eric Paulos. Sensr: evaluating a flexible framework for authoring mobile data-collection tools for citizen science. In *Proceedings of the 2013 conference on Computer supported cooperative work*, pages 1453–1462, 2013.

[38] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

[39] Reo Kuchida, Mayowa Olapade, Kevin Post, Ashfaq Hussain Ahmed, and Huber Flores. Assembling edge intelligence and decentralized infrastructure on command using llms. *Authorea Preprints*, 2024.

[40] Nicholas D Lane, Sourav Bhattacharya, Akhil Mathur, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar. Squeezing deep learning into mobile and embedded devices. *IEEE Pervasive Computing*, 16(3):82–88, 2017.

[41] Nicholas D Lane, Petko Georgiev, and Lorena Qendro. Deepear: robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In *Proceedings of the 2015 ACM international joint conference on pervasive and ubiquitous computing*, pages 283–294, 2015.

[42] Dawei Li, Theodoros Salonidis, Nirmit V Desai, and Mooi Choo Chuah. Deepcham: Collaborative edge-mediated adaptive deep learning for mobile object recognition. In *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 64–76. IEEE, 2016.

[43] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE signal processing magazine*, 37(3):50–60, 2020.

[44] Yige Li, Xixiang Lyu, Nodens Koren, Lingjuan Lyu, Bo Li, and Xingjun Ma. Neural attention distillation: Erasing backdoor triggers from deep neural networks. *arXiv preprint arXiv:2101.05930*, 2021.

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

[46] Yen-Wen Lin, Jie-Min Shen, and Hao-Jun Weng. Cloud-assisted gateway discovery for vehicular ad hoc networks. In *The 5th International Conference on New Trends in Information Science and Service Science*, volume 2, pages 237–240. IEEE, 2011.

[47] Shaoshan Liu, Jie Tang, Zhe Zhang, and Jean-Luc Gaudiot. Computer architectures for autonomous driving. *Computer*, 50(8):18–25, 2017.

[48] Daqin Luo, Chengjian Feng, Yuxuan Nong, and Yiqing Shen. Autom3l: An automated multimodal machine learning framework with large language models. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 8586–8594, 2024.

[49] Arnab Neelim Mazumder, Jian Meng, Hasib-Al Rashid, Utteja Kallakuri, Xin Zhang, Jae-Sun Seo, and Tinoosh Mohsenin. A survey on the optimization of neural network accelerators for micro-ai on-device inference. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 11(4):532–547, 2021.

[50] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[51] Brendan McMahan and Daniel Ramage. Federated learning: Collaborative machine learning without centralized training data. https://research.googleblog.com/2017/04/federated-learning-collaborative.html, 2017. Google AI Blog.

[52] Microsoft. Azure iot edge. https://azure.microsoft.com/en-us/products/iot-edge, 2024. Accessed: 2025-04-18.

[53] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. Darknetz: towards model privacy at the edge using trusted execution environments. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, pages 161–174, 2020.

[54] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[55] Ngoc Thi Nguyen, Agustin Zuniga, Huber Flores, Hyowon Lee, Simon Tangi Perrault, and Petteri Nurmi. Intelligent shifting cues: Increasing the awareness of multi-device interaction opportunities. In *Proceedings of the 29th ACM Conference on User Modeling, Adaptation and Personalization*, pages 213–223, 2021.

[56] Chengyi Nie, Rodrigo Fonseca, and Zhenhua Liu. Aladdin: Joint placement and scaling for slo-aware llm serving. *arXiv preprint arXiv:2405.06856*, 2024.

[57] Solmaz Niknam, Harpreet S Dhillon, and Jeffrey H Reed. Federated learning for wireless communications: Motivation, opportunities, and challenges. *IEEE Communications Magazine*, 58(6):46–51, 2020.

[58] Mayowa Olapade, Tarlan Hasanli, Abdul-Rasheed Ottun, Adeyinka Akintola, Mohan Liyanage, and Huber Flores. Pervasive chatbots: Investigating chatbot interventions for multi-device applications. In *Proceedings of the 32nd ACM Conference on User Modeling, Adaptation and Personalization*, pages 290–300, 2024.

[59] Kapil Patil and Bhavin Desai. Intelligent network optimization in cloud environments with generative ai and llms. 2024.

[60] Kevin Post, Reo Kuchida, Mayowa Olapade, Zhigang Yin, Petteri Nurmi, and Huber Flores. Contextllm: Meaningful context reasoning from multi-sensor and multi-device data using llms. In *Proceedings of ACM HOTMOBILE'25*. Association for Computing Machinery (ACM), 2025.

[61] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[62] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[63] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[64] Victor Sanh, Albert Webson, Colin Raffel, Stephen H Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, et al. Multitask prompted training enables zero-shot task generalization. *arXiv preprint arXiv:2110.08207*, 2021.

[65] 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.

[66] Nikolaos Schizas, Aristeidis Karras, Christos Karras, and Spyros Sioutas. Tinyml for ultra-low power ai and large scale iot deployments: A systematic review. *Future Internet*, 14(12):363, 2022.

[67] Kewei Sha, T Andrew Yang, Wei Wei, and Sadegh Davari. A survey of edge computing-based designs for iot security. *Digital Communications and Networks*, 6(2):195–202, 2020.

[68] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. "andromaly": a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, 38(1):161–190, 2012.

[69] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180, 2023.

[70] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.

[71] Shanmugasundaram Sivakumar. Performance optimization of large language models (llms) in web applications. *International Journal of Advanced Scientific Research*, 8:1077–1096, 2024.

[72] Statista Research Department. Internet of things (iot) connected devices worldwide 2019–2030. `https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/`, 2023. Accessed: 2025-04-18.

[73] Girish Suryanarayana, Mamadou H Diallo, Justin R Erenkrantz, and Richard N Taylor. Architectural support for trust models in decentralized applications. In *Proceedings of the 28th international conference on Software engineering*, pages 52–61, 2006.

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

[75] Andrew S Tanenbaum and Maarten Van Steen. *Distributed systems*. CreateSpace Independent Publishing Platform, 2017.

[76] Chandra Thapa, Pathum Chamikara Mahawaga Arachchige, Seyit Camtepe, and Lichao Sun. Splitfed: When federated learning meets split learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 8485–8493, 2022.

[77] Lorenzo Valerio, Andrea Passarella, and Marco Conti. A communication efficient distributed learning framework for smart environments. *Pervasive and Mobile Computing*, 41:46–68, 2017.

[78] Orkhan Valikhanli. Uav networks dos attacks detection using artificial intelligence based on weighted machine learning. *Results in Control and Optimization*, 16:100457, 2024.

[79] Niels Van Berkel, Denzil Ferreira, and Vassilis Kostakos. The experience sampling method on mobile devices. *ACM Computing Surveys (CSUR)*, 50(6):1–40, 2017.

[80] Blesson Varghese, Nan Wang, Sakil Barbhuiya, Peter Kilpatrick, and Dimitrios S Nikolopoulos. Challenges and opportunities in edge computing. In *2016 IEEE international conference on smart cloud (SmartCloud)*, pages 20–26. IEEE, 2016.

[81] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564*, 2018.

[82] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S Rellermeyer. A survey on distributed machine learning. *Acm computing surveys (csur)*, 53(2):1–33, 2020.

[83] Zhibo Wang, Jingjing Ma, Xue Wang, Jiahui Hu, Zhan Qin, and Kui Ren. Threats to training: A survey of poisoning attacks and defenses on machine learning systems. *ACM Computing Surveys*, 55(7):1–36, 2022.

[84] Pete Warden and Daniel Situnayake. *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*. O'Reilly Media, 2019.

[85] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

[86] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[87] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[88] Haojun Xia, Zhen Zheng, Yuchao Li, Donglin Zhuang, Zhongzhu Zhou, Xiafei Qiu, Yong Li, Wei Lin, and Shuaiwen Leon Song. Flash-llm: Enabling cost-effective and highly-efficient large generative model inference with unstructured sparsity. *arXiv preprint arXiv:2309.10285*, 2023.

[89] Cong Xie, Sanmi Koyejo, and Indranil Gupta. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934*, 2019.

[90] Tianwei Xing, Sandeep Singh Sandha, Bharathan Balaji, Supriyo Chakraborty, and Mani Srivastava. Enabling edge devices that learn from each other: Cross modal training for activity recognition. In *Proceedings of the 1st International Workshop on Edge Systems, Analytics and Networking*, pages 37–42, 2018.

[91] Dianlei Xu, Tong Li, Yong Li, Xiang Su, Sasu Tarkoma, Tao Jiang, Jon Crowcroft, and Pan Hui. Edge intelligence: Architectures, challenges, and applications. *arXiv preprint arXiv:2003.12172*, 2020.

[92] Dianlei Xu, Tong Li, Yong Li, Xiang Su, Sasu Tarkoma, Tao Jiang, Jon Crowcroft, and Pan Hui. Edge intelligence: Empowering intelligence to the edge of network. *Proceedings of the IEEE*, 109(11):1778–1837, 2021.

[93] Guowen Xu, Hongwei Li, Hao Ren, Jianfei Sun, Shengmin Xu, Jianting Ning, Haomiao Yang, Kan Yang, and Robert H Deng. Secure and verifiable inference in deep neural networks. In *Proceedings of the 36th Annual Computer Security Applications Conference*, pages 784–797, 2020.

[94] Mindy Yang and Gary Thung. Classification of trash for recyclability status. *CS229 project report*, 2016(1):3, 2016.

[95] Rui Ye, Wenhao Wang, Jingyi Chai, Dihan Li, Zexi Li, Yinda Xu, Yaxin Du, Yanfeng Wang, and Siheng Chen. Openfedllm: Training large language models on decentralized private data via federated learning. In *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*, pages 6137–6147, 2024.

[96] Liangqi Yuan, Ziran Wang, Lichao Sun, S Yu Philip, and Christopher G Brinton. Decentralized federated learning: A survey and perspective. *IEEE Internet of Things Journal*, 2024.

[97] Betul Yurdem, Murat Kuzlu, Mehmet Kemal Gullu, Ferhat Ozgur Catak, and Maliha Tabassum. Federated learning: Overview, strategies, applications, tools and future directions. *Heliyon*, 2024.

[98] Abe Zeid, Sarvesh Sundaram, Mohsen Moghaddam, Sagar Kamarthi, and Tucker Marion. Interoperability in smart manufacturing: Research challenges. *Machines*, 7(2):21, 2019.

[99] Xingzhou Zhang, Yifan Wang, Sidi Lu, Liangkai Liu, Weisong Shi, et al. Openei: An open framework for edge intelligence. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1840–1851. IEEE, 2019.

[100] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

I. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Reo Kuchida**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Edge Intelligence on Command,

   supervised by Mayowa Olapade, Huber Flores.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Reo Kuchida
15/5/2025