Technische Universität München
TUM School of Computation, Information and Technology

Efficient On-Device Learning in Resource-Constrained Systems: Adaptive Models, Data Selection, and Sparse Backpropagation

Marcus Florian Rüb

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung eines

Doktors der Ingenieurwissenschaften (Dr. Ing.)

genehmigten Dissertation.

Vorsitz: Prof. Dr. Ing. Hussam Amrouch

Prüfende der Dissertation:

1. Prof. Dr.-Ing. Daniel Müller-Gritschneder
2. Prof. Dr.-Ing Axel Sikora

Die Dissertation wurde am 08.04.2025 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 15.07.2025 angenommen.

Efficient On-Device Learning in Resource-Constrained Systems: Adaptive Models, Data Selection, and Sparse Backpropagation

ABSTRACT

Edge Machine Learning (Edge ML) represents a pivotal advancement in deploying machine learning capabilities directly on resource-constrained devices such as microcontrollers. While many existing approaches focus solely on inference, this thesis goes a step further by targeting **on-device training**. Training on the edge enables models to adapt to new data in real time without the need for external computation, but it poses significant challenges due to the severe constraints on memory, processing power, and energy availability. These constraints demand novel strategies for **data selection**, **efficient backpropagation**, and **adaptive learning**, which are traditionally designed for resource-rich environments.

This thesis introduces a comprehensive framework to enable on-device training by tackling three central challenges in Edge ML: (1) how to intelligently **select and store only the most informative data points**, (2) how to **train models efficiently under tight computational budgets using sparse backpropagation**, and (3) how to **adapt model complexity over time** to respond to changing input distributions. These components are tightly integrated to ensure that model performance is preserved while drastically reducing the resource footprint.

The research is structured around four guiding questions:

1. How can data selection be optimized in Edge ML environments?
2. What are effective incremental learning techniques for enhancing memory efficiency in Edge ML?
3. How can sparse backpropagation methods be improved to increase computational efficiency in Edge ML?
4. In what ways can adaptive learning models be designed to dynamically adjust to changing data and environmental conditions in Edge ML applications?

To answer these questions, the thesis contributes: (i) the **DRIP (Drop Unimportant Data Points)** algorithm, addressing the challenge of *data selection* through real-time, relevance-based filtering of input data; (ii) **novel incremental learning strategies** for *memory-efficient model updating*, enabling continual learning under storage constraints; and (iii) the **TinyProp** framework for *sparse backpropagation*, which

dynamically reduces computational effort during training while preserving model performance.

These methods are evaluated through theoretical analysis and rigorous experimentation across benchmark datasets such as MNIST, CIFAR-10, CIFAR-100, Speech Commands, Plant Disease, and HAR. The findings demonstrate that the integrated approach maintains near-baseline accuracy while drastically lowering resource consumption. For example, in the MNIST case, computation was reduced to 1% and memory usage to 22%—achieving **comparable performance at a fraction of the resource cost**.

In summary, this thesis contributes practical, efficient, and adaptable methodologies for **training neural networks directly on edge devices**. By maintaining performance with dramatically lower computational and memory requirements, these contributions pave the way for scalable, autonomous, and intelligent systems operating independently in real-world environments.

Contents

Chapter 1

Introduction

The rapid growth of data-driven technologies has elevated Machine Learning (ML) and Deep Learning (DL) as pivotal tools for solving complex problems across various domains. However, the deployment of these powerful models on resource-constrained devices, such as microcontrollers and other embedded systems, presents unique challenges. While many approaches focus on **inference**, where pre-trained models are deployed on devices to make predictions, this thesis emphasizes **on-device training**, enabling models to learn and adapt directly on the device itself. [1]

Inference vs. On-Device Training: Inference refers to the execution of a trained model on edge devices to process new data and generate predictions, often leveraging optimizations like pruning and quantization to fit the constraints of edge hardware. For example, a smart thermostat might use inference to predict room temperature based on sensor data. On-device training, on the other hand, allows models to be continuously updated or fine-tuned on new data generated by the device. This is critical for applications where data patterns change over time, such as personalized healthcare devices adapting to a patient's evolving condition or predictive maintenance systems that must account for wear and tear in machinery. [2]

The challenges of on-device training are significant. It requires efficient handling of Data Selection to minimize memory usage, computationally efficient methods like sparse backpropagation to update model parameters, and adaptive learning algorithms to respond to dynamic environments. Addressing these challenges is essential for making on-device training viable, especially on devices with limited computational resources, memory, and energy availability. [3]

In this chapter, we explore the necessity to bring ML and DL to the edge and how Edge Machine Learning (Edge ML) offers a solution to these challenges. The key concepts and workflows, as well as the relevance and challenges of on-device training are introduced. Lastly,Ioutline the research objectives, methodologies, and contributions of this thesis, setting the stage for an in-depth exploration of Edge ML.

1.1 Machine Learning, Deep Learning and the Wish to Bring it to the Edge

Machine Learning (ML) is a subset of artificial intelligence (AI) [4] that focuses on the development of algorithms and statistical models that enable computers to perform specific tasks without explicit instructions. These models are trained using vast amounts of data, allowing them to learn patterns, make decisions, and predict outcomes. ML techniques are broadly categorized into supervised learning, unsupervised learning, and reinforcement learning, each serving distinct purposes and applications. [5]

Deep Learning (DL) is a specialized branch of ML that involves neural networks with many layers, known as deep neural networks (DNNs). These networks are particularly effective for complex tasks such as image and speech recognition, natural language processing, and autonomous driving. Deep learning models have shown remarkable success due to their ability to learn hierarchical representations of data, capturing intricate patterns and features at multiple levels of abstraction. [6]

Bringing ML and DL to embedded devices, commonly referred to as Edge Machine Learning (Edge ML), addresses several critical needs and opens up new opportunities. One significant advantage is the proximity to the data source, which allows for processing data close to where it is generated. This proximity significantly reduces latency and improves real-time processing capabilities, crucial for delay-sensitive applications where geographical displacement between data-generating devices and cloud servers would otherwise cause detrimental delays.

Additionally, Edge ML reduces the dependence on cloud computing by leveraging the vast number of microcontroller units (MCUs) available. This approach alleviates the burden on cloud resources and decreases the frequency of data transmission to the cloud, saving bandwidth and energy. MCUs generally operate within the milliwatt range, allowing them to run on small batteries for extended periods, such as a coin cell battery lasting over a year. This energy efficiency makes Edge ML applications highly cost-effective compared to traditional edge devices.

Furthermore, by processing data locally, Edge ML helps preserve user privacy and enhance security. Local processing minimizes the risks associated with data breaches and vulnerabilities in cloud servers. The scalability and ubiquity of Edge ML are also notable, as the anticipated growth in the number of IoT devices, necessitates scalable solutions that can support a wide range of applications from healthcare to environmental monitoring.

Edge ML enables the optimization of idle computational resources available in the vast number of existing MCUs, ensuring efficient use of these devices for intelligent tasks without requiring significant additional infrastructure. This opens up new possibilities for always-on, inexpensive, battery-driven ML applications. Examples include voice-based assistants, where Edge ML models, such as Keyword Spotting algorithms, are already in use for activation features. [7]

1.2 Overview of Edge Machine Learning (Edge ML)

Edge Machine Learning (Edge ML) represents a significant advancement in the field of machine learning by bringing cognitive capabilities to resource-constrained devices such as microcontroller units (MCUs). The term "Edge ML" was initially coined by Warden et al. to describe machine learning applications that utilize miniaturized models for extremely energy-efficient inference [8]. Doyu et al. further refined this definition by highlighting the intersection of ML and constrained-IoT devices that operate without resource-rich operating systems such as Linux, Windows, or macOS [9]. A broad range of devices, including the Arduino UNO, ESP8266, and ATmega328, fall under the category of MCUs. Additional Edge ML-compatible boards can be found in various research endeavors and on the TensorFlow Lite website.

1.2.1 Definition of Edge ML

Edge ML refers to the deployment of machine learning models on resource-constrained devices like microcontrollers, which are characterized by their limited memory, processing power, and energy availability. The primary aim of Edge ML is to enable the execution of intelligent applications in environments where traditional machine learning models are not feasible due to these constraints.

Typical applications of Edge ML include voice recognition, anomaly detection, predictive maintenance, and sensor data processing. The key advantage of Edge ML is its ability to bring machine learning capabilities to small, power-efficient devices that are often embedded in larger systems or operate as standalone units in remote or inaccessible locations.

The roots of Edge ML stem from the advent of EdgeML, which reduces the need for continuous data transfer between devices and central servers, thus alleviating the infrastructural demands of conventional IoT. Edge ML is not intended to replace Fog or Cloud paradigms but rather to complement them, working in tandem to provide more comprehensive and efficient solutions. [6]

The scope of Edge ML is vast and interdisciplinary, encompassing computer science, electrical engineering, data science, and more. It plays a crucial role in the Internet of Things (IoT) ecosystem by enabling smart and autonomous decision-making at the network edge. By processing data locally, Edge ML devices can reduce latency, save bandwidth, and enhance privacy, as less sensitive data needs to be transmitted to the cloud or central servers.

1.2.2 Workflow of Edge ML

The conventional workflow for deploying Edge ML can be observed in Figure 1.1. This process is divided into three distinct phases: training, optimizing, and deploying. However, the architecture can be abstracted into traditional ML and Edge ML components. The former encapsulates data collection, algorithm selection, model training, and optimizing phases, which are de-facto conventional in traditional ML approaches. The latter specializes in Edge ML and consists of the model porting and deploying phases. [8]

The preliminary step in developing a Edge ML model is to select a relevant algorithm and collect the necessary data. This data can be sourced from precompiled data repositories or gathered in real-time using sensors. For instance, the authors of [10] used the Google Speech Commands repository to train and evaluate a keyword spotting algorithm, while [11] utilized a sensor-composed dataset to train a computer vision algorithm for self-driving micro-cars. Once the data is collected, the model training process can be conducted on a resource-rich device such as a server or personal computer. Contemporary ML frameworks like TensorFlow, PyTorch, and Scikit-Learn are typically employed for training.

Upon completion of model training, the next step is model optimization. This phase is critical for adapting the model to run efficiently on resource-constrained devices. Techniques such as pruning, knowledge distillation, quantization, and encoding are employed to achieve the desired requirements [12]. Pruning involves systematically removing parameters from a neural network to reduce its size without significantly affecting performance [13]. Knowledge Distillation (KD) involves training a smaller, optimized model (student model) to mimic the activations of a larger, pre-trained model (teacher model) [14]. Quantization reduces the numerical precision of a network's weights and activations, typically converting them from floating-point representations to integers, thus making the model more efficient in terms of speed and storage [15].

The final steps in a Edge ML solution are porting and deploying the optimized model. At this stage, the models are typically frozen, meaning their parameters are fixed, and the functions required for training, such as backpropagation, are excluded. This optimization minimizes resource consumption, enabling the models to perform only inference tasks, such as making predictions based on incoming data. While this approach is suitable for static environments, it limits the ability of models to learn and adapt to changes in data or operating conditions. Porting involves converting the model into a format that can be understood and executed by microcontroller units (MCUs). Typically, this involves translating the model into C/C++ code using Edge ML interpreters like TensorFlow Lite Micro [16]. These interpreters convert the model into a frozen graph, ideally represented as a C array, which can be embedded into the MCU [17].

Once ported, the model is deployed onto the MCU. The MCU performs inference on sensor data using the embedded ML model. This deployment allows for real-time processing and decision-making directly on the device, leveraging the optimized capabilities of Edge ML. A comprehensive survey of available optimizer and Edge ML compilers is discussed by Sanchez-Iborra et al. in [18].

This workflow ensures that Edge ML models are tailored to the constraints and re-

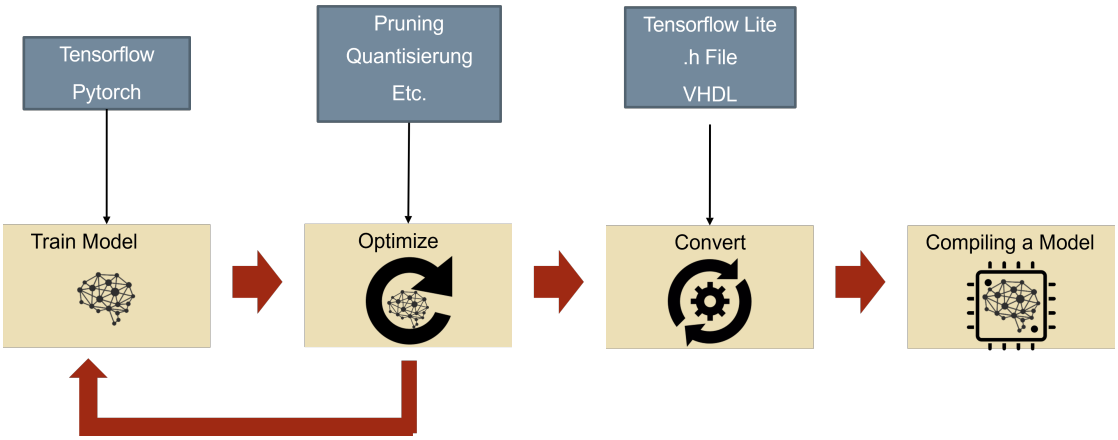

Figure 1.1: Workflow of Edge ML Model Deployment: The process begins with model training using frameworks like TensorFlow or PyTorch. The trained model is then optimized using techniques such as pruning and quantization to reduce its size and computational requirements. After optimization, the model is converted into a format suitable for deployment on resource-constrained devices, such as TensorFlow Lite or VHDL. Finally, the model is compiled and embedded into a microcontroller for on-device inference.

quirements of embedded devices, enabling them to operate efficiently and effectively in various real-world applications.

1.2.3 Deployment Methods of Edge ML

The widespread adoption of embedded solutions advocates continuous research in low-resource technologies that fall under the general criteria of Edge ML. A critical component of Edge ML solutions is the embedded inference mechanisms, where significant research efforts have been directed. It should be noted that models generated by popular ML frameworks such as TensorFlow [16], PyTorch [19], and Scikit-Learn [20] are unsuited for Edge ML in their raw form due to their substantial memory requirements. Consequently, multiple runtimes have been proposed to support ML/DL on microcontroller units (MCUs). However, three distinct methods have been identified for deploying Edge ML solutions: Manual Programming, Code Generators, and ML Interpreters [21].

Manual Programming yields the best results primarily due to the possibility of selectively optimizing all aspects. Despite the potential for producing highly optimized solutions, sharing knowledge is often limited because the optimizations made are typically obscured and confined to a select party. Moreover, the heterogeneity of MCUs hinders the capacity to unify these specialized Embedded ML approaches. From a functional standpoint, Edge ML solutions benefit in terms of performance from this approach at the cost of reproducibility and time, making such implementations less than ideal for non-domain experts, such as hobbyists. However, due to the absence of robust toolsets that allow models to be updated dynamically in real-time, manual programming often

becomes the only viable approach in scenarios where frequent or flexible updates to the model are required.

Code Generation tools are favored for their convenience, allowing an optimal solution to be reached without the need for manual programming. Many code generation tools, such as EdgeImpulse [22] and others, exploit AutoML to provide the necessary tools as Software-as-a-Service. Some tools, like [23], act as search engines for third-party Edge ML libraries. A drawback of code generation approaches is the fragmented market and proprietary toolsets and compilers from various vendors, making interoperability and portability a challenge [24].

Interpreters, in contrast, offer superior portability as the structure is similar regardless of the device. ML/DL models can be easily ported because a model's architecture is not tightly coupled to the framework. Additionally, models can be individually optimized and fine-tuned to suit specific needs and devices. Despite the convenience, this scheme comes with a slight overhead in performance and memory usage. The separation between model and system-level processes allows for greater generalizability, making it easier to introduce benchmarks and comparisons.

As interpreters can make embedded machine learning accessible to all, well-established technological companies are taking open-source initiatives to unify Edge ML. For instance, Google leads with TensorFlow Lite Micro (TFLM) [16], and Microsoft with the Embedded Learning Library (ELL) [25]. These frameworks play a crucial role in promoting the adoption of Edge ML by providing robust, community-supported platforms for development and deployment.

In summary, the deployment of Edge ML solutions can follow one of three primary methods: manual programming, code generation, and interpreters. Each method offers distinct advantages and trade-offs, from the high performance and specificity of manual programming to the convenience of code generation tools and the portability and generalizability of interpreters.

1.2.4 Characteristics of Embedded Systems

Embedded systems, often used interchangeably with microcontroller units (MCUs), exhibit distinct features that make them suitable for various intelligent edge applications. The advantages and disadvantages of such systems are discussed below.

Energy Efficiency

Embedded systems, particularly MCUs, are designed to operate with minimal energy consumption, which is one of their most notable advantages. Their ability to function efficiently with low power makes them well-suited for applications where energy resources are limited. This characteristic allows these systems to be powered by small batteries or even through alternative energy sources like solar power, vibrations, or thermal energy [18]. The combination of energy efficiency and compact size enables MCUs to be deployed in locations that lack consistent access to power, making them versatile for a wide range of uses. Additionally, many embedded systems are designed to further conserve energy by reducing the need for constant communication with external networks, enhancing their suitability for autonomous, low-maintenance applications [26]

Integration Possibilities

Advancements in high-throughput, energy-efficient architectures have enabled the miniaturization of System on Chips (SoCs), allowing sufficiently resourced processors to fit into devices with virtually limitless placement options. For example, [27] designed a sensor that can be implanted under the skin and powered by subcutaneous energy harvesting. This demonstrates the compactness and energy efficiency of such systems. MCUs are widely available and easily accessible, largely due to their low cost. This cost-effectiveness fosters use cases such as low-latency analysis and modeling of sensor signals from various industries, including manufacturing, agriculture, e-health, and entertainment [6]. As a result, industries are increasingly motivated to replace electro-mechanical or analog circuitry with software-defined alternatives [16]. The re-programmability and ubiquity of MCUs offer endless opportunities to develop highly dynamic intelligent systems.

Heterogeneity

The diversity within embedded devices contributes to their widespread adoption across various industries. However, this heterogeneity also presents a significant disadvantage: the market is heavily fragmented in terms of manufacturers and use cases. Consequently, countless devices with vastly different resource capacities are available, making benchmarking a challenging and tedious task.

In summary, the characteristics of embedded systems, including energy efficiency, flexible placement, cost-effectiveness, and heterogeneity, play a crucial role in their adoption and application in intelligent edge solutions. Despite the challenges, these features provide a strong foundation for deploying Edge ML models on MCUs, enabling advanced functionalities in resource-constrained environments. [28]

1.2.5 Key Attributes of Edge Devices for Machine Learning Applications

Previously,We discussed the distinctive features of embedded systems, highlighting their suitability for alternative processing at the edge. Coupling microcontroller units (MCUs) with cognitive capabilities can enable new application use cases and reveal previously unseen characteristics at the extreme edge, which will be discussed here.

Responsiveness

Most networks formed using low-powered embedded devices are categorized into Wireless Sensor Networks (WSNs) or Low Power and Lossy Networks (LLNs). Intermediate connections in LLNs are characterized by high packet loss rates, low bandwidth, and instability [29], making such networks impractical for delay-critical and connection-critical applications. Similarly, WSNs rely on a centralized primary node, causing delays in data processing based on the node's capabilities. A proven alternative is to process sensor data locally, significantly improving responsiveness. Enhanced responsiveness is crucial for future applications. For instance, offloading tasks in Augmented Reality (AR) glasses to the cloud, edge, or a neighboring device can cause delays, resulting in an unpleasant user experience. The necessity for always-on, battery-driven processing in AR demonstrates Edge ML as a suitable candidate [24].

Privacy

Ensuring data privacy and security is an inherent advantage of processing data on-premise. Localizing data processing eliminates the need to transmit raw data to or from a central entity, reducing the chances of privacy and security breaches. Additionally, duty cycling [30], which involves scheduling the power on/off states of a Radio Access Technology (RAT) for radio communication, can enhance security and device longevity. This ability to toggle the state of the RAT aligns well with concepts such as Collaborative Machine Learning, which focuses on improving a global model collectively. [12]

Energy Efficiency

One of the key benefits of Edge ML on MCUs is the significant reduction in energy consumption. Unlike traditional setups that rely heavily on cloud computing, which requires constant data transmission and high energy expenditure, ML on embedded devices minimizes the need for communication with external servers. This localized processing translates into lower energy demands, making it feasible to power devices using small batteries or energy-harvesting methods. The resulting energy efficiency extends the operational life of devices in the field, which is crucial for applications in remote or inaccessible locations. [31]

Scalability

The deployment of ML models on embedded devices allows for the scaling of intelligent applications across a vast number of devices. Given that billions of IoT devices are

expected to be in operation, Edge ML offers a scalable solution that distributes computational tasks across these devices, avoiding bottlenecks associated with centralized processing. This scalability also supports the proliferation of smart applications across diverse industries, from healthcare to agriculture, where real-time, localized decision-making is essential [32].

Cost-Efficiency

Integrating ML capabilities directly into MCUs can significantly reduce overall system costs. By leveraging the existing infrastructure of embedded devices, there is no need for expensive, high-power computing resources typically required for ML tasks. This cost-efficiency makes it possible to deploy advanced analytics and intelligent features even in low-budget or large-scale applications where traditional ML implementations would be prohibitively expensive [33].

Autonomy and Reliability

ML on embedded devices enhances system autonomy and reliability. Since decisions and computations are made locally, devices are less dependent on external networks, reducing potential points of failure. This autonomy is particularly beneficial in environments where network connectivity is unreliable or unavailable. Additionally, the reduced reliance on external resources means that these devices can continue to operate effectively even in challenging or isolated environments, ensuring consistent performance and reliability. [1]

Real-Time Processing and Low Latency

Edge ML on embedded devices excels in scenarios where real-time processing and low latency are critical. By conducting data processing and inference directly on the device, latency is minimized, allowing for immediate responses to sensor inputs. This capability is vital in applications such as autonomous vehicles, industrial automation, and real-time health monitoring, where delays could lead to suboptimal outcomes or even hazards. [34]

Customization and Flexibility

Embedded ML systems offer a high degree of customization and flexibility. Developers can tailor ML models specifically for the unique requirements of the device and application, optimizing for factors such as power consumption, processing speed, and memory usage. This flexibility allows for the creation of highly specialized applications that are fine-tuned to the constraints and demands of their operational environment. [35]

Security and Robustness

Another benefit of Edge ML is the increased robustness and security of the system. By keeping data and processing local, potential attack surfaces are reduced, as there is less

reliance on cloud-based services that can be vulnerable to breaches or attacks. Local processing ensures that sensitive data does not need to travel over public or insecure networks, further improving the security of the device. Additionally, the ability to run inference or training locally increases the robustness of the system in case of network outages or security threats, ensuring consistent operation even in disconnected or compromised environments [12].

Sustainability

Finally, the reduced energy consumption, extended operational life, and minimized reliance on high-power cloud servers contribute to the sustainability of ML applications on embedded devices. This approach aligns with broader environmental goals by decreasing the carbon footprint associated with data processing. The use of energy-efficient, long-lasting devices reduces electronic waste and helps to extend the life cycle of hardware deployed in the field, making Edge ML a more sustainable solution for modern IoT ecosystems [7].

In Conclusion, Edge ML-infused MCUs exhibit a range of advantageous characteristics, including enhanced responsiveness, privacy, energy efficiency, scalability, cost-effectiveness, autonomy, real-time processing, customization, security, and sustainability. These features make them suitable for various intelligent edge applications and highlight their potential to revolutionize the deployment of machine learning on resource-constrained devices.

1.3 Introduction to Edge ML On-Device Training

After exploring how Edge Machine Learning (Edge ML) enables the execution of machine learning models directly on resource-constrained devices in Section 1.2, we now turn our attention to an even more advanced capability—training these models directly on the device itself. On-device training refers to the process of training and updating machine learning models directly on the device where they are deployed, rather than relying on centralized servers. Unlike traditional machine learning models that are trained once on powerful servers and then deployed, on-device training allows models to continuously learn and adapt to new data in real-time, making them more responsive and capable of handling dynamic environments [35]. Traditional Edge ML models are typically frozen during deployment and are optimized solely for inference. While this ensures efficient performance on resource-constrained devices, it prevents the model from adapting to new data or environmental changes. In contrast, on-device training reintroduces the ability to update model parameters directly on the device, enabling continuous learning and dynamic adaptation. This capability is critical in applications where data distributions evolve over time, such as personalized healthcare or anomaly detection in industrial IoT.

1.3.1 Relevance of On-Device Training in Edge ML

The significance of on-device training in Edge ML is underscored by several critical factors. Firstly, on-device training optimizes the use of limited resources inherent in Edge ML devices, such as memory, computational power, and energy. By allowing models to adjust their complexity based on the available resources, on-device training ensures that these models operate efficiently without compromising their performance. This is particularly vital in environments where resource constraints are stringent [36].

Secondly, on-device training enhances the accuracy and efficiency of Edge ML models over time. This addresses a critical limitation of inference-only models, which, once deployed in their frozen state, are unable to learn from new data or adjust to changing environments. By enabling training directly on the device, on-device training bridges this gap and extends the utility of Edge ML models in dynamic and evolving scenarios. As these models continuously learn from new data, they can adapt to changing environments and improve their predictive capabilities. This adaptability is crucial for applications that require real-time responses to new types of data, such as anomaly detection in industrial IoT or personalized health monitoring [3].

Another key benefit of on-device training is its ability to reduce dependency on constant communication with a central server. By training and updating models locally on the device, it minimizes the need for data transmission to and from the cloud, thereby saving bandwidth and enhancing data privacy. This local processing is particularly important in applications where data privacy is a concern, as it reduces the risk of data breaches during transmission [37].

On-device training also addresses the challenge of model robustness in Edge ML. In dynamic environments where input data and conditions can change rapidly, it is essential for models to be able to update themselves in response to these changes. On-device training provides the mechanisms necessary for this continual adaptation, helping to maintain the model's accuracy and reliability over time [3].

Moreover, on-device training is well-suited for incremental learning scenarios, where a model gradually learns from new data while retaining previously acquired knowledge. This approach is especially relevant in Edge ML, where the constraints of the device may make it infeasible to frequently retrain the model from scratch [38].

In summary, on-device training is a foundational element in the advancement of Edge ML, enabling models to function effectively within the unique constraints and demands of Edge, resource-limited devices [1]. While many Edge ML implementations have traditionally focused on static inference, the potential benefits of on-device training are increasingly recognized. The ability to deploy MCUs in hard-to-reach locations (e.g., implanted sensors or remote machinery) makes on-device training an attractive option, as it allows models to learn from new data points or update wirelessly, thereby avoiding the need for frequent extraction, retraining, and redeployment.

Additionally, on-device training can help address the issue of model drift, where a model's performance degrades over time due to changes in the underlying data distribution. This phenomenon, often referred to as Concept Drift, occurs when the relationship between inputs and target variables evolves, as defined by Gama et al. [39]. For example,

a recommendation system that performs well in one season may become less accurate as consumer behavior changes in another season. On-device training enables models to adjust to these shifts in real-time, thereby maintaining their relevance and effectiveness.

1.3.2 Challenges in On-Device Training

Despite these advantages, on-device learning entails three main technical challenges. Firstly, there is a contradiction between hardware resource availability and demand. IoT devices typically have limited hardware resources, such as kilobytes or megabytes of memory. However, machine learning model training requires considerable hardware resources, including computing and memory. Model training can be described as an optimization problem where the goal is to minimize the loss, which measures the difference between the model output and the ground truth. During the training process, ML methods tune model parameters by following the gradient direction. Gradient-based training normally applies backpropagation algorithms to compute the gradient, which store a large amount of intermediate loss and output of hidden layers, consuming tens of gigabytes of memory or more. This contradiction makes on-device training challenging. [40]

Secondly, there is high heterogeneity in IoT devices. IoT devices are diverse and heterogeneous, covering a wide spectrum in terms of capability, from severely constrained microcontrollers to single-board computers with decent hardware, such as Raspberry Pi. This diversity makes it challenging to propose a single, generally applicable solution for all devices. Thirdly, there is limited existing work specifically targeting model training optimization for resource-constrained devices. Most existing optimization algorithms aim to achieve higher accuracy or faster convergence, with few focusing on enabling model training in resource-contained devices. [1]

Addressing these challenges is crucial for advancing the field of Edge ML and realizing its potential in various applications. Researchers must continue to develop innovative methods and solutions to overcome these limitations and enable effective on-device training for Edge ML.

1.4 Objectives and Research Questions of the Thesis

To systematically address the research questions, this thesis follows a structured framework that connects research challenges, novel contributions, and corresponding publications 1.1:

- **Q (Research Questions)**: The research questions define the core challenges in Edge ML addressed in this thesis, including data selection, incremental learning, sparse backpropagation, and adaptive learning. They serve as the foundation for the investigation and guide the development of novel methodologies.
- **C (Contributions)**: Each research question is tackled through specific contributions, introducing innovative approaches such as the DRIP algorithm for efficient

data selection, incremental learning methods for optimizing memory usage, and TinyProp for improving computational efficiency. These contributions enhance the feasibility of on-device training in resource-constrained environments.

- **P (Publications)**: The research findings and proposed methodologies are validated through rigorous experimental analysis and presented in peer-reviewed publications. Each publication is directly linked to one or more research questions and contributions, ensuring that the results are reproducible and contribute to advancing the field of Edge ML.

This structured alignment of research questions, contributions, and publications ensures a cohesive approach to tackling Edge ML challenges, leading to practical and impactful advancements in the field.

Table 1.1: Mapping of Research Questions (Q), Contributions (C), and Publications (P)

Q#	**Research Question**	**Contribution (C#)**	**Publication (P#)**
Q1	How can Data Selection be optimized in Edge ML environments?	C1	P2
Q2	What are effective incremental learning techniques for enhancing memory efficiency in Edge ML?	C2	P3
Q3	How can sparse backpropagation methods be improved to increase computational efficiency in Edge ML?	C3	P4
Q4	In what ways can adaptive learning models be designed to dynamically adjust to changing data and environmental conditions in Edge ML applications?	C4	P1–P4

1.4.1 Thesis Objectives

The primary goals of this thesis in the realm of Edge Machine Learning (Edge ML) are multifaceted, reflecting the broad scope and profound impact of this field. The objectives are as follows:

- **Advancement of Adaptive Learning Techniques:** Explore and develop novel methods that allow Edge ML models to dynamically adjust their behavior based on changing environments and data patterns.
- **Optimization of Data Selection and Management:** Develop strategies to efficiently handle and process data on resource-constrained devices, addressing the challenges associated with Data Selection and management in Edge ML.

- **Enhancement of Model Training and Updating Processes:** Improve the processes involved in training and updating Edge ML models, focusing on incremental and sparse learning techniques that enable effective model training and fine-tuning on-device.

Achieving these goals will not only contribute to the academic understanding of Edge ML but also pave the way for its broader application and impact in the real world.

1.4.2 Research Questions and Methodological Approaches

This thesis addresses several key research questions critical to advancing Edge Machine Learning (Edge ML). The research methods employed to answer these questions are outlined below, incorporating Method, Algorithm Development, Experimental Evaluation, Performance Metrics Analysis, and Comparative Studies.

1. **Q1: How can Data Selection be optimized in Edge ML environments?** (**C1**, **P2**)
   - **Method:** Introduce the novel algorithm DRIP (Drop Unimportant Data Points) algorithm to optimize Data Selection and management in resource-constrained environments.
   - **Algorithm Development:** Develop and implement the DRIP algorithm to prioritize and select the most relevant data points for processing and analysis in Edge ML.
   - **Experimental Evaluation:** Conduct experiments using benchmark datasets (e.g., MNIST, CIFAR-10, CIFAR-100, Speech Commands, Plant Disease, HAR) to assess the efficiency of the DRIP algorithm.
   - **Performance Metrics Analysis:** Measure and analyze key metrics such as accuracy, computational effort, and memory usage to demonstrate the efficiency of the Data Selection process.
   - **Comparative Studies:** Compare the DRIP algorithm with existing Data Selection methods to benchmark its performance and identify advantages and limitations.
2. **Q2: What are effective incremental learning techniques for enhancing memory efficiency in Edge ML?** (**C2**, **P3**)
   - **Method:** Investigate incremental learning techniques to optimize memory usage while maintaining model accuracy and performance in Edge ML environments.
   - **Algorithm Development:** Develop incremental learning methods to enable efficient model training and updating on Edge devices with memory constraints.
   - **Experimental Evaluation:** Conduct empirical studies on datasets to assess the effectiveness of the proposed incremental learning techniques.

- **Performance Metrics Analysis:** Analyze memory usage, accuracy, and efficiency metrics to evaluate the performance of the incremental learning techniques.
- **Comparative Studies:** Benchmark the developed incremental learning methods against existing approaches to highlight their improvements and limitations.

3. **Q3: How can sparse backpropagation methods be improved to increase computational efficiency in Edge ML?** (**C3**, **P4**)
   - **Method:** Explore sparse backpropagation methods like TinyPropv2 to reduce the computational burden during on-device model training.
   - **Algorithm Development:** Refine and enhance the TinyProp method to optimize computational efficiency in Edge ML environments.
   - **Experimental Evaluation:** Conduct experiments to validate the computational efficiency improvements offered by the enhanced sparse backpropagation techniques.
   - **Performance Metrics Analysis:** Measure and analyze key metrics such as computational effort and model accuracy to assess the efficiency of sparse backpropagation.
   - **Comparative Studies:** Compare the TinyPropv2 method with existing sparse backpropagation techniques to highlight its improvements and areas for future research.
4. **Q4: In what ways can adaptive learning models be designed to dynamically adjust to changing data and environmental conditions in Edge ML applications?** (**C4**, **P1-P4**)
   - **Method:** Design adaptive learning models that can dynamically adjust to changing conditions in Edge ML applications.
   - **Algorithm Development:** Develop adaptive learning models capable of real-time adjustment based on dynamic data and environmental conditions.
   - **Experimental Evaluation:** Validate the adaptability and robustness of the developed models through case studies and real-world deployments.
   - **Performance Metrics Analysis:** Measure and analyze how well the adaptive learning models maintain accuracy and efficiency under changing conditions.
   - **Comparative Studies:** Compare the adaptive learning models with existing approaches to highlight their effectiveness and potential for further development.

Addressing these research questions will provide valuable insights and contributions to the field of Edge ML, driving innovation and practical applications in this rapidly evolving domain.

1.5 Summary of Contributions of the Thesis

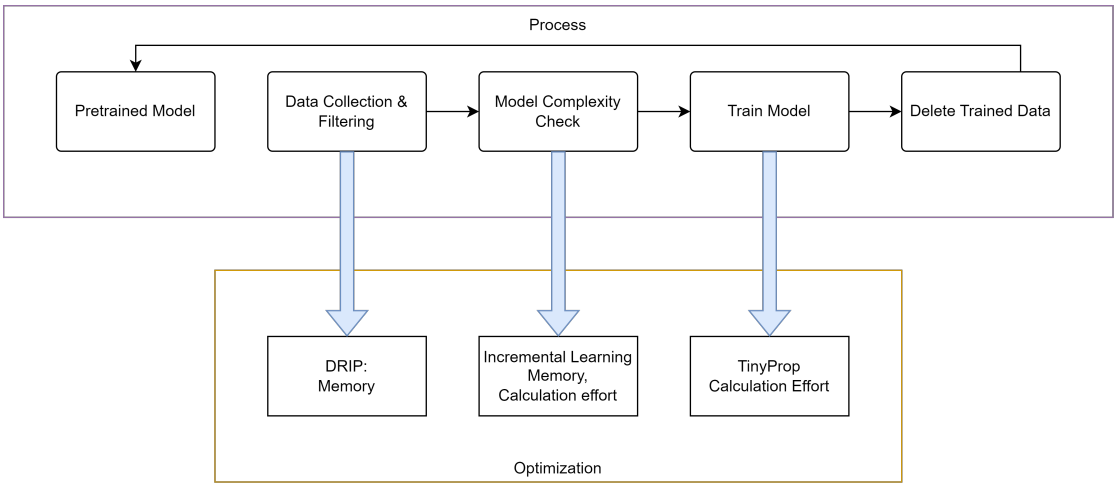

Figure 1.2: Schematic of the On-Device Neural Network Training Optimization: Showcasing a looped process that integrates a Pretrained Model, Data Collection & Filtering with DRIP for memory efficiency, Model Complexity evaluation for computational adaptability, and TinyPropv2-optimized Training, followed by data pruning to streamline continuous learning.

This thesis makes several innovative contributions to the field of Edge Machine Learning (Edge ML), which are summarized below:

- **C1: Innovative Techniques for Data Selection and Management (Q1, P2):** The introduction of the DRIP algorithm for optimizing Data Selection and management in Edge ML environments, enabling efficient processing of data on resource-constrained devices and enhancing model performance while reducing computational overhead.
- **C2: Advancements in Incremental Learning (Q2, P3):** Development of new incremental learning methods that demonstrate improved memory efficiency, making them well-suited for on-device learning scenarios by optimizing model updates without significant memory overhead.
- **C3: Enhancement of Sparse Backpropagation Methods (Q3, P4):** Contributions to the advancement of sparse backpropagation techniques, particularly through the newly developed TinyProp method, significantly improving computational efficiency during model training and making it viable for resource-limited edge devices.

- **C4: Development of Adaptive Learning Models (Q4, P1 - P4):** Design and implementation of adaptive learning models that dynamically adjust to varying data and environmental conditions, exhibiting enhanced robustness and flexibility crucial for the dynamic nature of Edge ML applications.

Overall, the contributions of this thesis represent a substantial advancement in Edge ML, offering practical, efficient, and robust solutions for deploying machine learning on edge, resource-constrained devices.

1.6 Publication Overview

The research conducted in this thesis has resulted in significant contributions to the field of Edge Machine Learning (Edge ML), culminating in several publications and presentations at esteemed journals and conferences. Each publication corresponds to key chapters of this thesis and addresses one or more of the research questions posed in the earlier sections. Below is an overview of these publications:

1. **P1: A Practical View on Training Neural Networks in the Edge:** This paper, presented at the 17th IFAC Conference on Programmable Devices and Embedded Systems (PDES) 2022 in Sarajevo, provides a broad literature review on the state of training neural networks in resource-constrained environments. While this work does not contribute directly to any specific research question or contribution, it serves as the foundational work that informed the development of the thesis. This publication corresponds to the general literature review efforts undertaken in the thesis. [1]

2. **P2: DRIP: Drop Unimportant Data Points - Enhancing Machine Learning Efficiency with Grad-CAM-Based Real-Time Data Prioritization:** This paper, derived from Chapter 3 and related to **Q1** and **C1**, was presented in 2025 IJCNN IEEE. It focuses on the development of the DRIP algorithm to optimize Data Selection in Edge ML environments. The research demonstrates the use of Grad-CAM for strategic Data Selection, achieving significant storage savings and model performance enhancements through real-time data prioritization. [41]

3. **P3: A Continual and Incremental Approach for Edge ML On-device Training Using Dataset Distillation and Model Size Adaption:** Based on the findings in Chapter 4 and addressing **Q2** and **C2**, this paper was presented at the 7th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS). The research introduces novel incremental learning techniques for Edge ML, focusing on memory-efficient model updates. The publication highlights the development of scalable and efficient incremental learning algorithms tailored for embedded devices and demonstrates their effectiveness across multiple datasets. [42]

4. **P4: Advancing On-Device Neural Network Training with TinyPropv2: Dynamic, Sparse, and Efficient Backpropagation:** This paper, presented at the WCCI 2024 IJCNN IEEE, builds upon the TinyProp algorithm and is derived from Chapter 5. Addressing **Q3**, and **C3**, it introduces TinyPropv2, a refined version of sparse backpropagation that dynamically adjusts the level of sparsity during on-device training. The paper highlights significant reductions in computational effort while maintaining high accuracy across various datasets. [40]

Collectively, all the publications (**P1**–**P4**) contribute to addressing **Q4** and **C4**, highlighting the advancements made in the area of computational efficiency and sparse backpropagation for Edge ML.

1.7 Organization of the Thesis

This thesis is structured into seven chapters, each addressing key research questions and contributions to advancing Edge Machine Learning (Edge ML).

- **Chapter 1: Introduction to Edge ML** This chapter introduces the concept of Edge ML, its significance, and the core objectives of this research. It outlines the critical challenges faced by resource-constrained devices in implementing machine learning and sets the stage for the research questions that will be addressed throughout the thesis. This chapter links to the research questions **Q1**–**Q4** and corresponding contributions **C1**–**C4**.

- **Chapter 2: Literature Review** Chapter 2 provides a comprehensive literature review on current trends and challenges in Edge ML, particularly focusing on on-device training, adaptive machine learning, data point selection, incremental learning, and efficient backpropagation techniques. This chapter reviews key methods and algorithms that serve as the foundation for addressing **Q1**, **Q2**, **Q3**, and **Q4**, as well as contributions **C1**, **C2**, **C3**, and **C4**. It is supported by **P1**.

- **Chapter 3: Data Selection Optimization in Edge ML** Chapter 3 addresses **Q1** and introduces the DRIP (Drop Unimportant Data Points) algorithm, which optimizes Data Selection for Edge ML environments. It includes detailed experimental evaluations and performance analysis of DRIP's impact on resource management, contributing to **C1**. The corresponding publication for this chapter is **P2**.

- **Chapter 4: Incremental Learning for Memory Efficiency in Edge ML** This chapter is dedicated to **Q2**, focusing on incremental learning techniques that enhance memory efficiency. It presents the development of novel methods that allow models to be incrementally updated in resource-constrained settings. The contributions here include **C2**, with experimental validation and performance analysis, supported by the publication **P3**.

- **Chapter 5: Sparse Backpropagation for Computational Efficiency in Edge ML** Chapter 5 addresses **Q3** by introducing and advancing the TinyPropv2 algorithm for sparse backpropagation. This technique reduces computational overhead during on-device training. The contributions made here are encapsulated in **C3**, with comprehensive experimental evaluation, and published as **P4**.
- **Chapter 6: Integrative Analysis and Comparative Study** In this chapter, an integrative analysis of the methodologies developed in Chapters 3, 4, and 5 is performed. The comparative study evaluates the performance, memory efficiency, and computational gains of the proposed techniques, linking together **Q1**–**Q3** to answer **Q4** and contributions **C4**. The results highlight the advantages and limitations of each approach across various use cases.
- **Chapter 7: Conclusion and Future Work** Chapter 7 summarizes the key findings of the thesis, with an emphasis on the implications for the field of Edge ML. It reflects on how the contributions—**C1** (Data Selection), **C2** (incremental learning), **C3** (sparse backpropagation), and **C4** (adaptive learning)—drive innovation in on-device training. It also discusses potential future research directions, aiming to further optimize machine learning for edge devices.

Chapter 2

Literature Review

In this chapter, the critical aspects of training neural networks in resource-constrained environments are examined through current trends, challenges, and innovative solutions 1. A thorough review of on-device training, Data Selection, incremental learning, and efficient backpropagation is provided, highlighting recent advancements and practical implementations.

2.1 On-Device Training

In this section, the various aspects of on-device training for neural networks are explored, emphasizing its significance and recent advancements. The section explores current trends, challenges, and practical case studies, showcasing how on-device training can be effectively implemented across different application domains.

2.1.1 Current Trends in On-Device Training

On-device training of neural networks has gained significant traction due to the growing demand for real-time data processing and privacy-preserving machine learning [2]. Unlike traditional methods that rely heavily on cloud-based servers for model training, on-device training leverages the computational capabilities of edge devices, such as smartphones, IoT devices, and microcontrollers. This paradigm shift is driven by advancements in hardware. Specialized AI accelerators like Google's Edge TPU [43] and NVIDIA's Jetson[44] series provide the necessary computational power within constrained environments. Additionally, recent developments in model optimization techniques, such as quantization and pruning, have enabled the deployment of smaller, yet highly efficient models. These models can be trained directly on these devices [45].

A notable trend in on-device training is the adoption of federated learning, where models are trained collaboratively across multiple devices while keeping the data localized. This approach not only enhances privacy by avoiding the need to transfer sensitive data but also reduces the communication overhead associated with centralized training. Techniques such as federated averaging (FedAvg) allow the aggregation of locally trained models into a global model, which is then redistributed to the devices for further training [46]. Moreover, incremental and continual learning strategies are being integrated into on-device training frameworks to allow models to adapt to new data without retraining

from scratch. This is particularly beneficial in dynamic environments where data evolves over time, ensuring that the models remain up-to-date and relevant [47].

Another emerging trend is the use of transfer learning and lightweight architectures tailored for on-device training. Transfer learning leverages pre-trained models as a starting point, significantly reducing the amount of data and computational resources required for training on new tasks. Architectures such as MobileNet [48], SqueezeNet [49], and EfficientNet [50] are designed to be compact and efficient, making them suitable for deployment and training on resource-constrained devices. These models are often coupled with optimization techniques like knowledge distillation [51]. In this approach, a smaller model (student) is trained to mimic the behavior of a larger, pre-trained model (teacher), further enhancing training efficiency. Collectively, these trends are paving the way for more capable and intelligent edge devices, enabling a wide range of applications from personalized health monitoring to smart home automation [52].

2.1.2 Case Studies in On-Device Training

Several case studies illustrate the practical implementation and benefits of on-device training across different application domains. One prominent example is the use of on-device training in personalized health monitoring systems. Researchers have developed wearable devices equipped with sensors and on-device machine learning capabilities to continuously monitor physiological signals such as heart rate, temperature, and activity levels. These systems use incremental learning algorithms to update health models in real-time based on individual user data, providing personalized health insights and early detection of potential health issues. This approach not only enhances privacy by keeping sensitive health data on the device but also ensures timely and relevant health monitoring without relying on cloud-based analysis [53].

Another significant case study is in the field of autonomous vehicles, where on-device training enables real-time adaptation to changing driving conditions. Autonomous vehicles are equipped with numerous sensors and cameras that generate vast amounts of data. Training models directly on these vehicles allows them to learn and adapt to new environments, road conditions, and driving patterns without the latency associated with cloud-based updates. For instance, a study demonstrated the efficacy of using edge computing units integrated with GPUs to perform continuous learning and adaptation in self-driving cars. The vehicles could improve their navigation systems in real-time, handling new obstacles and optimizing routes more efficiently [53].

A third notable case study is the deployment of on-device training in smart home systems. Smart home devices, such as intelligent thermostats, security cameras, and voice assistants, can benefit significantly from on-device training to improve their performance and user experience. For example, smart thermostats can learn the preferences and schedules of household members by continuously updating their models based on usage patterns and environmental changes. Security cameras with on-device training capabilities can improve their accuracy in detecting unusual activities by learning from real-time video feeds [54]. This enhancement in detection helps boost home security without the need for constant data transmission to the cloud. These case studies demon-

strate the versatility and effectiveness of on-device training in various fields, highlighting its potential to revolutionize how intelligent systems operate in resource-constrained environments [55].

2.1.3 Challenges and Solutions of On-Device Training

On-device training faces several significant challenges, primarily related to the limited computational resources, memory constraints, and energy efficiency of edge devices. Unlike cloud servers, which offer abundant processing power and storage, edge devices such as smartphones and IoT devices have constrained hardware capabilities. This limitation poses a challenge for running complex neural network training algorithms that require intensive computational resources and large amounts of memory. Additionally, the energy consumption associated with on-device training can be substantial, affecting the battery life of portable devices and the sustainability of IoT applications [2].

To address these challenges, researchers have developed several innovative solutions aimed at optimizing the efficiency of on-device training. One such solution is model compression, which includes techniques like pruning, quantization, and low-rank factorization. Pruning involves removing redundant weights from the neural network, thereby reducing the model size and computational load. Quantization reduces the precision of the weights and activations from floating-point to lower-bit representations, significantly decreasing the memory footprint and computational requirements. Low-rank factorization approximates the weight matrices with lower-rank representations, enabling faster computations and reduced memory usage. These techniques collectively help in fitting large models into the limited resources of edge devices without significantly compromising performance [56].

Another effective solution is the development of specialized training algorithms and frameworks that are tailored for resource-constrained environments. For instance, federated learning frameworks are designed to distribute the training process across multiple edge devices. This setup allows each device to perform local updates on the model using its own data. The locally updated models are then aggregated to form a global model, minimizing the need for data transfer and central computation [46]. Moreover, incremental learning algorithms enable models to update continuously as new data arrives, rather than retraining from scratch, thus saving computational resources [57]. Adaptive learning rate techniques and energy-efficient hardware accelerators, such as Google's Edge TPU and NVIDIA's Jetson, further enhance the feasibility of on-device training. They achieve this by optimizing power consumption and improving training speed. These solutions are critical in overcoming the inherent limitations of edge devices, making on-device training a viable option for real-world applications [58].

2.2 Data Selection

In this section, the focus is on the critical process of Data Selection in machine learning, particularly within the constraints of on-device training. The discussion encompasses current trends, the challenges faced, and the innovative solutions developed. It also includes illustrative case studies highlighting the practical applications and benefits of advanced Data Selection techniques.

2.2.1 Current Trends in Data Selection

Data Selection has emerged as a critical area of focus in machine learning, particularly for on-device training where computational and memory resources are limited. One of the current trends in this domain is the use of active learning techniques. These techniques aim to improve model performance by selectively choosing the most informative data points for training. Active learning algorithms, such as uncertainty sampling and query-by-committee, prioritize data points that are expected to reduce the model's uncertainty. By doing so, these algorithms ensure that the model is trained on the most valuable data, leading to faster convergence and improved accuracy with fewer training examples [59].

Another notable trend in Data Selection is the incorporation of coreset selection methods. Coreset algorithms create a smaller, representative subset of the original dataset that retains the essential characteristics and statistical properties of the full dataset. Techniques like K-Center and greedy algorithms are commonly used to identify and retain the most representative data points. This approach significantly reduces the amount of data required for training, thereby minimizing memory usage and computational overhead. The effectiveness of coreset selection has been demonstrated in various applications, including computer vision and natural language processing, where large datasets are commonplace [60].

In addition to active learning and coreset selection, the use of explainability and interpretability tools is becoming increasingly popular in Data Selection. Notably, Gradient-weighted Class Activation Mapping (Grad-CAM) is one such tool that is gaining widespread adoption. Grad-CAM helps visualize the regions of an input image that are important for the model's predictions. This enables researchers to identify and select data points that contribute most to the model's learning process. This approach not only enhances the efficiency of the training process but also provides insights into the model's decision-making, leading to more transparent and trustworthy AI systems. These trends collectively aim to optimize the training process by focusing on the most impactful data points, thereby enabling efficient and effective on-device learning [61].

Strumbelj and Kononenko [62] brought forth Data Shapley as a method to quantify the importance of individual data points within a dataset. By evaluating each point's contribution to the model's overall performance, Data Shapley offers guidance on Data Selection for both training and deployment. Despite its potential, Data Shapley is computationally intensive, particularly for large datasets, and operates offline, restricting real-time data importance assessments [63]. Such challenges necessitate continued research to harness its full potential in machine learning tasks.

Data Selection in Machine Learning: 'Ingredients', Strategies, and Open Challenges by Sim et al. [64] provides a comprehensive survey on Data Selection in machine learning, elucidating its "ingredients" and properties. While the authors present an encompassing view of the topic, they do not introduce a novel algorithm. In contrast, my approach pioneers in leveraging Grad-CAM for online Data Selection.

The paper from Wang and Jia [65] emphasizes the robustness of Data Selection, advocating for the Banzhaf value from cooperative game theory. The distinct direction of my work lies in the incorporation of Grad-CAM for online decision-making, providing a fresh perspective in Data Selection.

The work from Xu et al. [66] focused on the health domain and the pricing from data, this research introduces the Selection And Pricing mechanism called VAP mechanism for online Data Selection. While my method also operates online, it uniquely integrates Grad-CAM for decision-making, thus enriching the Data Selection landscape, especially when considering data point significance for the entire dataset.

2.2.2 Case Studies in Data Selection

Several case studies have demonstrated the efficacy of advanced Data Selection techniques in improving machine learning model performance, especially in resource-constrained environments. One notable case study is the application of active learning in medical image analysis. Researchers used uncertainty sampling to select the most informative medical images for training a neural network to detect diseases in radiographs. By focusing on images where the model was most uncertain, they significantly reduced the amount of labeled data required for training while maintaining high diagnostic accuracy. This approach not only minimized the labeling effort but also ensured that the model learned from the most challenging and informative cases. As a result, the model's generalization to new, unseen images was improved [67].

Another important case study involves the use of coreset selection in large-scale visual recognition tasks. In a project aimed at developing efficient object recognition models for autonomous drones, researchers employed K-Center clustering [68]. This technique was used to select a representative subset of training images from a vast dataset. This method allowed them to train models that were both lightweight and accurate, suitable for real-time processing on drones with limited computational resources. The selected coresets maintained the diversity and complexity of the original dataset, ensuring robust performance across various environmental conditions and object appearances. This study highlighted the practical benefits of coreset selection in reducing training times and memory usage without compromising the quality of the trained models [69].

A third case study explored the use of Grad-CAM [61] for enhancing Data Selection in the context of natural language processing (NLP) [70]. Researchers developed a sentiment analysis model for social media data. They used Grad-CAM to visualize and understand which parts of the text were most influential in the model's predictions. By selecting Data that highlighted diverse and critical aspects of sentiment, they were able to construct a highly informative training set. This approach improved the model's ability to accurately classify sentiments in varied and complex text inputs while reducing the

overall data requirements. The success of this case study demonstrated how explainability tools like Grad-CAM can be effectively integrated into the Data Selection process. This integration leads to more efficient and transparent machine learning models [71].

2.2.3 Challenges and Solutions of Data Selection

One of the primary challenges in Data Selection is identifying the most informative and representative data points from large datasets without introducing bias or overlooking important information. The process of selecting these data points can be computationally intensive, particularly when dealing with high-dimensional data or datasets with complex structures. Moreover, ensuring that the selected data points accurately represent the overall data distribution is critical to prevent model performance degradation. This challenge is exacerbated in dynamic environments where data characteristics continuously evolve, requiring frequent updates to the selected data points [72].

To address these challenges, researchers have developed several innovative solutions. One such solution is the use of advanced active learning techniques that leverage uncertainty estimation and diversity sampling. Uncertainty estimation methods, like Bayesian neural networks, provide a measure of the model's confidence in its predictions. This allows for the selection of data points where the model is most uncertain. Diversity sampling, on the other hand, ensures that the selected data points cover a broad range of the data distribution, preventing overfitting to specific patterns. Combining these approaches helps create a balanced and informative training set that improves model performance while reducing computational requirements [73].

Another effective solution is the implementation of coreset selection algorithms. These algorithms efficiently identify and retain a small subset of the data, preserving the essential properties of the full dataset. Techniques like K-Center clustering and greedy selection algorithms are designed to optimize the selection process by focusing on data points that are most representative of the overall dataset. Additionally, recent advancements in explainable AI, such as the use of Grad-CAM for visualizing important features, have provided valuable tools for enhancing Data Selection. These tools allow researchers to understand and interpret the model's decision-making process, leading to more informed and effective selection of data points. By integrating these solutions, researchers can overcome the challenges of Data Selection, enabling efficient and robust on-device training in resource-constrained environments [74].

2.2.4 Contributions Beyond Current Approaches

This thesis builds upon the existing advancements in Data Selection by introducing the **DRIP (Drop Unimportant Data Points)** algorithm, which uniquely integrates Grad-CAM for real-time, online data prioritization. While prior research has focused on techniques like active learning, coreset selection, and explainability tools individually, the DRIP algorithm combines these methods to address the specific constraints and requirements of on-device training.

- **Real-time Data Selection with Explainability:** Unlike traditional Data Shapley methods, which are computationally intensive and operate offline, the DRIP algorithm leverages Grad-CAM to perform real-time assessments of data importance. This enables immediate decision-making about which data points to retain or discard, significantly reducing computational overhead and memory requirements during on-device training.
- **Enhanced Efficiency and Storage Optimization:** DRIP reduces storage needs by up to 39% without compromising model performance, as demonstrated in experiments across diverse datasets, including MNIST [75], CIFAR-10 [76], and Speech Commands [77]. This improvement surpasses the efficiency of coreset selection and active learning techniques, which typically focus on reducing training set size without explicitly optimizing storage for edge devices.
- **Online Adaptability and Resource-Constrained Suitability:** Existing approaches such as Data Shapley and K-Center clustering often assume access to centralized computing resources for data importance calculations. In contrast, DRIP is specifically tailored for the constraints of resource-limited devices, allowing for on-device execution and adaptation without relying on external computational support.

By combining real-time prioritization, explainability, and dynamic adaptability, the DRIP algorithm addresses critical gaps in the state-of-the-art. It provides a practical and efficient solution for Data Selection in on-device training scenarios, paving the way for more robust and resource-aware machine learning applications at the edge.

2.3 Incremental Learning

In this section, the focus is on incremental learning, a method that enables models to continuously learn from new data without needing to retrain from scratch. The section explores current trends, challenges, and innovative solutions in incremental learning, and presents case studies that demonstrate its practical applications across various domains. This capability is particularly important for embedded devices, where memory and processing constraints make frequent full retraining infeasible, yet continuous adaptation to new data is essential for maintaining performance in dynamic environments.

2.3.1 Current Trends in Incremental Learning

Incremental learning, which allows models to learn continuously from new data without retraining from scratch, has become a focal point in machine learning. This approach is especially vital for on-device training in resource-constrained environments. One current trend is the development of algorithms that mitigate catastrophic forgetting, a common issue where models lose previously learned information upon learning new data. Techniques such as Elastic Weight Consolidation (EWC) [78] and Synaptic Intelligence (SI) [79] have been widely adopted. These methods work by selectively updating network

weights and protecting those important for previously learned tasks. EWC, for instance, penalizes changes to weights that are essential for old tasks, ensuring that new learning does not overwrite established knowledge [80]. Additionally, methods like ILOVA use replay-based approaches to counter catastrophic forgetting with minimal memory usage, making them suitable for resource-constrained devices [81]. This highlights the need for memory-efficient techniques, like the one presented in this work, which prioritize both memory usage and learning efficiency.

Another significant trend is the integration of memory-augmented neural networks (MANNs) and rehearsal strategies [82]. MANNs incorporate an external memory component that helps retain information about past tasks, enabling the model to recall and utilize this information when learning new tasks. Rehearsal strategies involve storing a subset of past data and mixing it with new data during training. This approach ensures that the model is periodically exposed to old data, thus reinforcing previously learned information. Techniques like GEM (Gradient Episodic Memory) use rehearsal strategies to balance learning new information while retaining old knowledge. This makes them particularly effective for incremental learning scenarios in dynamic environments [83]. Solutions like HARNet use deep ensembles for on-device incremental learning, providing efficiency in human activity recognition but lack the strong memory optimization that your method introduces [84].

The use of lightweight, modular architectures is also gaining traction in incremental learning. Architectures like Progressive Neural Networks and Dynamic Expandable Networks (DEN) allow the model to grow and adapt over time by adding new neurons or layers for new tasks [85]. These architectures retain the existing structure for previously learned tasks. This modular approach not only facilitates the integration of new knowledge without interference but also maintains computational efficiency, which is essential for on-device training. Additionally, recent advancements in meta-learning and continual learning frameworks, such as Model-Agnostic Meta-Learning (MAML), are being applied to incremental learning [86]. These frameworks enable models to rapidly adapt to new tasks with minimal updates, leveraging prior knowledge to accelerate the learning process. Together, these trends are advancing the capabilities of incremental learning, making it more effective and feasible for deployment in real-world applications where continuous adaptation and learning are essential.

While architectures like IL4IoT [87] provide efficient incremental learning frameworks for IoT, they still lack solutions for efficient memory usage during on-device training. The method addresses this by focusing on improving memory efficiency without compromising learning performance, making it especially useful for real-time applications.

2.3.2 Case Studies in Incremental Learning

One notable case study in incremental learning is the application of EWC in reinforcement learning for robotics. Researchers deployed EWC in robotic systems to enable them to learn multiple tasks sequentially without forgetting previously learned behaviors. For instance, a robot initially trained to navigate through an obstacle course could later learn to pick up and place objects without losing its navigation capabilities. EWC

was essential in maintaining performance across different tasks by preserving the weights important for the navigation task. It allowed the model to adapt to the object manipulation task. This approach demonstrated significant improvements in multi-task learning for robotics, enabling continuous and adaptive learning in real-world environments [88].

Another significant case study involves the use of Generative Replay in NLP for continual learning. In this study, researchers aimed to develop a chatbot capable of learning new conversational topics over time while retaining the ability to engage in previously learned discussions. By incorporating a generative model that produced synthetic data representative of past conversations, the chatbot could be incrementally trained on new dialogue datasets. This generative replay mechanism effectively mitigated catastrophic forgetting, allowing the chatbot to maintain high performance across both old and new conversation topics. The study highlighted the effectiveness of generative replay in applications requiring the integration of new information while preserving existing knowledge [89].

A third case study focused on using PNNs in autonomous driving systems. PNNs were employed to enable an autonomous vehicle to learn new driving scenarios, such as different weather conditions and varying traffic patterns. This allows the vehicle to adapt to new conditions without losing proficiency in previously learned scenarios. In this study, the vehicle was initially trained on a set of baseline driving conditions and then exposed to new conditions sequentially. By adding new columns of neurons for each new scenario while preserving the existing columns, PNNs allowed the vehicle to retain its performance in earlier conditions. This approach enabled the vehicle to seamlessly adapt to new ones. This approach not only improved the vehicle's adaptability and robustness but also reduced the need for extensive retraining, making it more practical for real-world deployment. These case studies collectively illustrate the diverse applications and effectiveness of incremental learning techniques in various domains. They highlight the potential of these techniques to enable continuous learning and adaptation in complex, dynamic environments [90].

2.3.3 Challenges and Solutions of Incremental Learning

Incremental learning faces several significant challenges, with one of the most prominent being catastrophic forgetting. This phenomenon occurs when a neural network, trained sequentially on different tasks, tends to overwrite previously learned information with new data, leading to a rapid decline in performance on earlier tasks. Addressing this challenge requires methods that can balance the retention of old knowledge while integrating new information. One effective solution is EWC, which penalizes changes to weights critical for old tasks by adding a regularization term to the loss function. Another approach is SI, which tracks the importance of weights over time and adjusts updates accordingly, ensuring that important weights are preserved during new learning phases [91]. However, even with solutions like Train++, which enhances self-learning on IoT devices, there is still a need for more memory-efficient approaches that scale better in resource-constrained environments [92].

Another challenge in incremental learning is managing the computational and memory

resources required to store and process historical data. Traditional rehearsal methods, which involve retaining a subset of past data and replaying it during training, can be memory-intensive and impractical for resource-constrained devices. To mitigate this, researchers have developed techniques like Generative Replay, where a generative model is trained alongside the main model to generate synthetic data representative of past experiences. This synthetic data can then be used during training to prevent forgetting. Additionally, efficient memory management strategies have been proposed to optimize resource usage while maintaining model performance. One such strategy is using a fixed-size memory buffer that selectively retains the most informative samples [93]. Approaches like Incremental Learning on Chip demonstrate the feasibility of integrating incremental learning on hardware, but further optimization for memory efficiency is necessary to handle larger datasets or more complex tasks [38].

2.3.4 Contributions Beyond Current Approaches

This thesis advances the field of incremental learning by introducing a novel memory-efficient approach tailored for on-device training, addressing key limitations of existing methods. Unlike traditional incremental learning techniques, which often rely on extensive memory resources or fixed architectures, this research presents solutions designed specifically for resource-constrained environments.

- **Memory-Efficient Incremental Learning:** While existing approaches such as Elastic Weight Consolidation (EWC) [78] and Synaptic Intelligence (SI) [79] mitigate catastrophic forgetting, they typically require significant memory resources to store weight importance metrics. This thesis introduces an adaptive model-sizing mechanism that dynamically adjusts model complexity based on available memory, optimizing resource usage without sacrificing performance.
- **Integration of Dataset Distillation:** In contrast to traditional rehearsal strategies that retain subsets of historical data, the proposed method leverages dataset distillation to create compact, representative synthetic datasets. This approach drastically reduces memory requirements while maintaining the model's ability to recall and integrate previously learned knowledge.
- **Adaptation to Dynamic Environments:** The research extends traditional memory-augmented neural networks (MANNs) [82] by integrating dynamic adaptation mechanisms. These mechanisms enable the model to prioritize learning from new data while preserving critical knowledge from prior tasks, ensuring robustness in environments where data patterns evolve over time.
- **Scalable and Modular Architectures:** Building on the concepts of Progressive Neural Networks (PNNs) [94] and Dynamic Expandable Networks (DENs) [85], this thesis introduces scalable architectures that allow for efficient task-specific expansions. The proposed approach maintains high performance while minimizing resource consumption, providing a practical solution for deployment on edge devices.

By addressing the challenges of catastrophic forgetting, memory limitations, and computational inefficiencies, this thesis goes beyond current approaches to provide a comprehensive, resource-aware framework for incremental learning. These advancements enable the deployment of robust, adaptive machine learning models in resource-constrained environments, paving the way for more effective on-device training in real-world applications.

2.4 Efficient Backpropagation

In this section, the focus is on efficient backpropagation techniques essential for on-device training in resource-constrained environments. The section explores current trends, addresses significant challenges, and presents case studies that demonstrate the practical application and benefits of these advanced backpropagation methods.

2.4.1 Current Trends in Efficient Backpropagation

Efficient backpropagation has become a critical focus in machine learning, particularly for on-device training where computational resources are limited. One of the current trends in this area is the development of sparse backpropagation techniques, which aim to reduce the number of gradient updates during the training process. Methods like top-k sparsification and gradient pruning selectively update only the most significant weights, thereby reducing the computational load. For instance, top-k sparsification retains only the top k% of the gradients with the highest magnitude, effectively compressing the gradient information and minimizing the number of updates. This approach significantly reduces the memory and processing power required for training, making it suitable for resource-constrained devices [95].

Another prominent trend is the use of low-precision arithmetic in backpropagation. Techniques such as quantization and fixed-point arithmetic reduce the precision of computations, allowing for faster and more energy-efficient processing. By representing weights and activations with fewer bits, these methods decrease the memory footprint and computational complexity of neural networks. Quantization-aware training, where models are trained with low-precision arithmetic from the outset, has shown to maintain accuracy comparable to full-precision models while offering substantial improvements in efficiency. This trend is particularly relevant for deploying deep learning models on edge devices, where power and memory constraints are critical considerations [96].

Additionally, the integration of specialized hardware accelerators is revolutionizing efficient backpropagation. Devices like Google's Tensor Processing Units (TPUs) [43] and NVIDIA's Tensor Cores [44] are designed to optimize the performance of deep learning computations, including backpropagation. These accelerators leverage parallel processing and optimized memory access patterns to speed up training and reduce energy consumption. Furthermore, advances in neuromorphic computing, such as Intel's Loihi chip [97], mimic the brain's neural architecture to perform efficient learning with low power consumption. These hardware innovations, combined with algorithmic advancements, are driving significant improvements in the efficiency of backpropagation. This progress

enables more effective on-device training and broader deployment of AI applications in real-world settings [98].

2.4.2 Challenges and Solutions of Efficient Backpropagation

One of the primary challenges in efficient backpropagation is the high computational and memory demands associated with training deep neural networks. Backpropagation involves numerous matrix multiplications and gradient calculations, which require significant processing power and memory bandwidth [40]. For resource-constrained devices, such as smartphones and IoT sensors, these requirements can be prohibitive. To address this, researchers have developed sparse backpropagation techniques that reduce the number of active gradients and weights during training. Methods such as top-k sparsification and gradient pruning ensure that only the most significant updates are performed. This approach decreases the computational load and memory usage without substantially compromising model accuracy [99].

Another significant challenge is maintaining the precision and stability of computations in resource-constrained environments. Low-precision arithmetic, while beneficial for reducing computational and memory requirements, can introduce quantization errors that affect the stability and performance of the training process. Solutions to this issue include quantization-aware training. This approach incorporates the effects of low-precision arithmetic during the training phase itself, allowing the model to learn robust representations despite reduced precision. Additionally, techniques like mixed-precision training, which combines high-precision and low-precision computations, help to balance efficiency and accuracy. By strategically using higher precision for critical parts of the network and lower precision for less sensitive components, these methods ensure stable and efficient training [100].

Efficient backpropagation also faces challenges related to hardware limitations. Standard processors are not optimized for the highly parallel nature of neural network training, leading to inefficiencies in computation and energy usage. To overcome this, the development of specialized hardware accelerators, such as GPUs, TPUs, and neuromorphic chips, has become a critical solution. These accelerators are designed to handle the parallel computations of neural networks more efficiently, significantly speeding up the training process and reducing energy consumption. Furthermore, software frameworks like TensorFlow Lite [16] and PyTorch Mobile [19] have been optimized to leverage these hardware accelerators effectively, providing tools and libraries that facilitate efficient on-device training. By addressing these hardware-related challenges, researchers are enabling the deployment of complex neural networks on a wide range of devices, from mobile phones to embedded systems [101].

2.4.3 Case Studies in Efficient Backpropagation

One notable case study in efficient backpropagation involves the use of sparse backpropagation techniques in real-time speech recognition systems. Researchers at Google implemented top-k sparsification to enhance the performance of their speech recognition

models on mobile devices [77]. By selectively updating only the most significant gradients during training, they were able to reduce the computational load and memory usage significantly. This approach allowed the speech recognition models to be trained and updated directly on users' smartphones, providing real-time, personalized speech recognition without the need for extensive cloud resources. The implementation demonstrated that sparse backpropagation could maintain high accuracy while operating within the constraints of mobile hardware, making it a viable solution for on-device learning [102].

Another significant case study is the application of low-precision arithmetic in image classification tasks on edge devices. NVIDIA researchers utilized quantization and mixed-precision training techniques to optimize the training of convolutional neural networks (CNNs) on their Jetson Nano [44] platform, which is designed for edge computing. By using 8-bit and 16-bit arithmetic instead of the standard 32-bit floating-point operations, they achieved substantial reductions in both memory footprint and power consumption. The quantization-aware training ensured that the models retained high classification accuracy despite the lower precision. This case study highlighted the effectiveness of low-precision arithmetic in making complex image recognition tasks feasible on resource-constrained edge devices. It paved the way for more widespread adoption of AI at the edge [103].

A third case study focused on the integration of specialized hardware accelerators in autonomous drone navigation. Researchers at MIT developed a custom neuromorphic chip [97] to accelerate the backpropagation process for training neural networks used in drone navigation. The chip, designed to mimic the brain's neural architecture, allowed for efficient parallel processing and significantly reduced power consumption. By leveraging this specialized hardware, the drones could perform real-time learning and adaptation to changing environments and obstacles. The neuromorphic chip enabled continuous training and updating of the navigation models without relying on external computational resources. This case study demonstrated the potential of neuromorphic computing to revolutionize on-device learning for autonomous systems, making them more adaptable and efficient in real-world scenarios [104].

2.4.4 Contributions Beyond Current Approaches

This thesis introduces significant advancements in efficient backpropagation tailored for on-device training, addressing the limitations of existing techniques in resource-constrained environments. The contributions focus on optimizing computational efficiency, reducing memory overhead, and enabling real-time adaptability for edge devices.

- **Dynamic Sparse Backpropagation:** While traditional sparse backpropagation methods like top-k sparsification focus on fixed sparsity thresholds [95], this thesis introduces a dynamic approach that adjusts the level of sparsity during training. By dynamically identifying and updating the most critical gradients, the proposed method minimizes computational overhead while maintaining high model accuracy, making it highly adaptable for diverse edge device constraints.

- **Efficient Backpropagation on Heterogeneous Hardware:** Building on advancements in specialized hardware, such as neuromorphic chips and Tensor Cores [44], this thesis provides optimized algorithms that leverage hardware-specific features to enhance the efficiency of backpropagation. The proposed methods ensure compatibility with heterogeneous hardware platforms, facilitating broader adoption of efficient training techniques in real-world applications.
- **Enhanced Real-Time Learning Capabilities:** By integrating sparse backpropagation with real-time data prioritization techniques, this thesis enables edge devices to perform continuous training and adaptation. This capability is particularly valuable for applications requiring immediate responses to dynamic environments, such as autonomous navigation and personalized user interactions.

These contributions go beyond the current state of the art by addressing the dual challenges of computational efficiency and adaptability in resource-constrained environments. The proposed methods demonstrate significant improvements in energy efficiency, memory utilization, and training speed, paving the way for more effective deployment of machine learning models on edge devices.

2.5 Summary

In this chapter, We explored the critical aspects of training neural networks in resource-constrained environments. The review highlighted the current trends, challenges, and innovative solutions in the areas of on-device training, Data Selection, incremental learning, and efficient backpropagation. On-device training emerged as a promising approach for enabling real-time adaptability and privacy preservation by minimizing the reliance on cloud servers. This method is complemented by advancements in Data Selection techniques, such as active learning and coreset selection, which optimize model performance while reducing computational and memory overhead.

Furthermore, incremental learning techniques allow models to continuously adapt to new data without the need for retraining, addressing the challenge of catastrophic forgetting. Efficient backpropagation methods, including sparse backpropagation and low-precision arithmetic, were shown to mitigate the resource constraints of edge devices, facilitating the deployment of more complex neural networks.

The case studies presented in this chapter demonstrated the practical implementation of these techniques across various domains, from healthcare to autonomous systems. Collectively, these developments illustrate the ongoing progress toward enabling robust, adaptive machine learning directly on edge devices, opening up new possibilities for intelligent applications in real-world, resource-constrained environments.

Chapter 3

Novel Method to Optimize Memory and Computational Resources via Strategic Data Selection in Edge ML

In this chapter, strategies for optimizing memory and computational resources in Edge ML through strategic Data Selection are examined. Techniques such as the DRIP algorithm for data retention and various experimental validations are discussed. These methods demonstrate how efficient Data Selection can enhance the performance of Edge ML model training on resource-constrained devices.

3.1 Fundamentals of Data Selection in ML

Data Selection in machine learning is pivotal for model performance and efficiency. It involves choosing the most relevant and informative data from available datasets for training. Effective Data Selection enhances model accuracy, reduces training time, and is particularly crucial in applications with computational and storage limitations.

3.1.1 Principles of Data Selection

Data Selection in machine learning involves choosing subsets of data that are most effective for training models. This process is governed by principles that ensure the chosen data contributes meaningfully to the learning process. Mathematically, these principles can be formalized and quantified, providing a framework for effective Data Selection.

- **Relevance:** Mathematically, relevance can be quantified by measuring how well data points align with the specific learning task. This involves analyzing the correlation or mutual information between data features and the target variable. For a dataset D with features X and target Y, relevance R can be quantified as:

$$R = I(X; Y|D) \tag{3.1}$$

  where I denotes the mutual information, indicating how much knowing the features reduces uncertainty about the target.

- **Representativeness:** Representativeness ensures that the selected data captures the underlying distribution of the full dataset. It can be assessed by comparing statistical properties, such as mean μ and variance σ^2, of the selected subset S against the entire dataset D. A representative subset satisfies:

$$\mu_S \approx \mu_D \quad \text{and} \quad \sigma_S^2 \approx \sigma_D^2 \tag{3.2}$$

- **Diversity:** Diversity in Data Selection ensures a wide range of features and scenarios are included, preventing model overfitting. Mathematically, this can be ensured by maximizing the pairwise distances or dissimilarities among the selected data points. For a subset S, diversity Dv can be expressed as:

$$Dv(S) = \sum_{i,j \in S, i \neq j} d(x_i, x_j) \tag{3.3}$$

  where $d(x_i, x_j)$ measures the distance or dissimilarity between data points x_i and x_j.

These mathematical formulations provide a foundation for implementing Data Selection algorithms that are aligned with the core principles of relevance, representativeness, and diversity, thus ensuring effective and efficient learning in machine learning models.

3.1.2 Relevance in Edge ML

In Edge Machine Learning (Edge ML), data selection plays a critical role in ensuring the efficient operation of models deployed on resource-constrained devices. Unlike general-purpose machine learning systems, Edge ML applications are bounded by strict limitations in memory, computational power, and energy availability. As a result, the inclusion of data that does not contribute significantly to model learning may lead to unnecessary computational and storage overheads.

From a systems perspective, selecting only the most informative and representative data points can mitigate resource consumption while preserving or enhancing model performance. This is particularly important in continuous or on-device training scenarios, where real-time adaptation must be achieved within the bounds of constrained hardware. Efficient data selection, therefore, is not solely a matter of improving accuracy but also of meeting the operational constraints that define Edge ML environments.

- **Efficient Use of Limited Resources:** In Edge ML, resources are at a premium. Efficient Data Selection allows for the most informative and relevant data to be processed, thereby optimizing the use of limited memory and processing power. This careful selection reduces the model size and complexity, making it feasible for deployment on Edge devices.

- **Enhanced Model Performance:** By focusing on data that provides the most value for model training, Edge ML models can achieve higher accuracy and robustness. This is particularly crucial in applications where the model needs to make reliable predictions based on real-time data.

- **Reduced Energy Consumption:** Efficient Data Selection directly impacts the energy consumption of Edge ML devices. Processing less but more relevant data can significantly lower the energy requirements, which is vital for battery-powered or energy-harvesting devices.
- **Improved Data Privacy:** With more data processing occurring on the device itself, the need to transmit sensitive data to the cloud for processing can be reduced. This approach enhances user privacy, a growing concern in an increasingly connected world.
- **Real-World Applicability:** In practical scenarios, such as in IoT and wearable devices, the ability to make quick, accurate decisions based on selected data is invaluable. Efficient Data Selection empowers these devices to function autonomously and react promptly to environmental changes or user inputs.

Therefore, in Edge ML, the art and science of Data Selection become fundamental not just for achieving high model performance but also for ensuring the viability and sustainability of machine learning in the most resource-constrained environments.

3.2 Datapoint Selection through DRIP

Embedded systems in Edge Machine Learning (Edge ML) operate under severe constraints in computational power, memory, and energy consumption. Given these limitations, optimizing data retention is critical to ensure that only the most informative samples are stored for future model updates. Traditional data selection approaches, such as uncertainty sampling or diversity-driven heuristics, either fail to account for the model's internal representation or are computationally prohibitive for real-time applications. 2.2

To bridge this gap, we propose leveraging Gradient-weighted Class Activation Mapping (Grad-CAM) [61] as a principled method for assessing the importance of individual data points. While Grad-CAM is traditionally used for explainability, its capacity to highlight decision-critical regions provides a means to quantify the relevance of input samples. In this section, we outline the motivation for using Grad-CAM in data selection, demonstrating how its interpretability can be systematically transformed into a decision criterion for retention or discard.

3.2.1 From Model Decision-Making to Data Selection

The Model's Perspective Matters In an ideal setting, data selection should be guided by the model's own decision-making process rather than external heuristics. This perspective shifts the focus from generic statistical measures of data utility to model-aware selection criteria. Specifically, we require a mechanism to measure how much the model "cares" about a given data point—i.e., how strongly it contributes to the model's internal feature representations.

While existing methods such as Data Shapley [62] or influence functions attempt to assess data importance, they are computationally intensive and require access to the entire dataset. These approaches are unsuitable for real-time selection in embedded applications. Instead, a lightweight yet informative metric is needed—one that directly reflects the model's learning dynamics without imposing significant computational overhead.

Grad-CAM as a Proxy for Model Attention Grad-CAM [61] offers a model-centric method for evaluating the relevance of individual data points by visualizing which regions of an input were most responsible for a given prediction. Unlike general feature importance techniques, which evaluate input features independently of the model's learned decision boundaries, Grad-CAM ties importance directly to the network's internal representations. This makes it particularly suitable for evaluating whether a data point provides meaningful learning signals.

For a given input sample, Grad-CAM generates a heatmap that highlights the spatial regions that most influenced the model's decision. If a model strongly activates for specific patterns, this suggests that the data point is integral to the model's feature space. Conversely, weak or diffuse activations indicate that the sample contributes little to the learned representation. This distinction provides a natural foundation for determining which data points should be retained.

From Explainability to Data Selection The transition from model interpretability to data selection requires a method to quantify the importance of each data point systematically. Our hypothesis is as follows:

- If an input sample produces strong, localized activations in its Grad-CAM heatmap, it carries essential information for the model and should be retained.
- If an input sample results in weak, scattered, or diffuse activations across all target classes, it is less relevant to the model's learning process and can be discarded without degrading performance.

This insight allows us to formulate a retention criterion based on a quantitative measure derived from Grad-CAM heatmaps. Specifically, we introduce the DRIP Score (Drop Unimportant Data Points Score), which aggregates the activation strength of a data point's Grad-CAM heatmap into a single numerical value. The DRIP Score serves as an automatic selection threshold, determining whether a data point is stored or discarded.

3.2.2 Leveraging Grad-CAM for Data Selection

Grad-CAM [61] emerges as an effective tool for data selection in embedded devices due to its ability to offer interpretable, lightweight, and versatile insights into a model's decision-making process. This capability aligns well with the stringent resource constraints of Edge ML. The key reasons for its suitability are outlined below:

1. Grad-CAM Reflects Decision-Critical Features: Grad-CAM highlights the regions of an input that the model deems most influential for its predictions. These highlighted regions correspond to the decision-critical features that the model has learned from the training dataset.

- **Focused and Clear Heatmaps:** When Grad-CAM produces focused heatmaps, it indicates that the model has successfully identified meaningful patterns in the data. For instance, in image classification tasks, these regions might correspond to distinct objects or features relevant to the target class.
- **Diffuse or Irrelevant Activations:** Conversely, if the heatmap is scattered or focuses on irrelevant regions, it can signify:
  - The data point is redundant, contributing little to the training process.
  - The model struggles to generalize this data point, possibly due to gaps in the training dataset.

This ability to distinguish between informative and uninformative data points makes Grad-CAM an invaluable tool for assessing data utility.

2. The Model as a Proxy for Dataset Understanding: A trained neural network inherently reflects the structure and information contained in its training dataset. Grad-CAM enables a direct examination of this representation, offering insights into how well the model "understands" specific data points.

- **Localized and Strong Activations:** If Grad-CAM outputs are concentrated on meaningful regions, it suggests that the dataset provides sufficient and representative information for the model to perform effectively.
- **Weak or Scattered Activations:** On the other hand, weak or diffuse heatmaps may indicate noisy, underrepresented, or poorly labeled data, highlighting areas where the dataset could benefit from refinement.

By probing a model's internal representation through Grad-CAM, We gain a powerful lens to evaluate the dataset's quality and identify opportunities for improvement.

3. Explainability for Embedded Applications: Explainability is a critical requirement for deploying machine learning models on embedded devices. These systems often operate in resource-constrained environments where computational resources must be allocated judiciously. Grad-CAM provides a transparent view of a model's decision-making process, enabling informed decisions about which data points are worth retaining for further training.

4. Lightweight and Computationally Feasible: Generating Grad-CAM heatmaps involves relatively low computational overhead, making it well-suited for embedded applications. Unlike other interpretability methods that may require substantial processing power, Grad-CAM strikes a balance between insight and efficiency, allowing it to run on devices with limited energy and processing capabilities. As discussed in Section 3.2.8, the computational overhead in the production phase is minimal.

5. Versatility Across Data Modalities: Grad-CAM is not limited to visual data; it has been successfully applied to audio signals, time-series data, and even structured sensor inputs. This versatility ensures that Grad-CAM can be seamlessly integrated into various Edge ML scenarios, making it a robust and adaptable choice for data selection in real-world applications.

3.2.3 From Insights to Decisions: A Metric for Data Selection

While Grad-CAM provides detailed insights into the importance of specific regions of an input, effectively utilizing these insights requires a systematic metric for making binary retention decisions. To address this need, We introduce a scoring mechanism based on Grad-CAM heatmaps, detailed in the subsequent sections. This mechanism computes a numerical score for each data point, enabling us to decide whether to retain or discard it based on its contribution to the model's learning. By translating interpretability into actionable decisions, this approach ensures that embedded devices can efficiently manage limited resources while maintaining high performance.

3.2.4 Grad-CAM: Visualizing Model Decisions

Introduced by Selvaraju et al.[61], Grad-CAM stands as a seminal technique for visual interpretation within convolutional neural networks (CNNs). The method offers a mechanism to determine regions within an input image that influence the model's predictions. By examining gradients of the target class concerning the model's final convolutional layer, Grad-CAM elucidates the focus areas in an image. Such insights are valuable for model diagnosis and enhancement. Beyond images, Jahmunah et al. [105] demonstrated Grad-CAM's applicability to ECG data, underscoring its versatility.

Gradient-weighted Class Activation Mapping (Grad-CAM) is a visualization technique used to illustrate which regions of an input image contribute the most to a neural network's prediction. It provides insights into the decision-making process of convolutional neural networks (CNNs) by producing a heatmap that highlights the influential regions of the input.

The core idea behind Grad-CAM is to compute the gradient of the output score for a target class (before the softmax operation) with respect to the feature maps of a convolutional layer. These gradients serve as weights to produce a weighted combination of the feature maps, resulting in a coarse heatmap of the same size as the feature maps.

The steps to compute the Grad-CAM heatmap are as follows:

1. Let y^c be the score for class c (before the softmax). Compute the gradients of y^c with respect to the feature maps A of a convolutional layer:

$$\frac{\partial y^c}{\partial A^k} \tag{3.4}$$

where k is the index of the feature map.

2. Perform Global Average Pooling on the gradients to obtain the weights α_k^c for each feature map:

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k} \tag{3.5}$$

where Z is the number of elements in feature map A^k, and i, j are spatial indices.

3. Compute the weighted combination of the feature maps to obtain the raw Grad-CAM heatmap $L^c_{\text{Grad-CAM}}$:

$$L^c_{\text{Grad-CAM}} = \sum_k \alpha_k^c A^k \tag{3.6}$$

4. Apply the ReLU activation function to the raw heatmap to obtain the final Grad-CAM heatmap:

$$L^c_{\text{Grad-CAM}} = \max(0, L^c_{\text{Grad-CAM}}) \tag{3.7}$$

The resulting heatmap $L^c_{\text{Grad-CAM}}$ can be resized to the dimensions of the input image and overlaid on it to visualize the regions that influenced the prediction for class c. This visualization provides an intuitive understanding of the model's decision-making process, highlighting areas of the image that were pivotal in determining the output class.

3.2.5 DRIP Algorithm

In the realm of machine learning and especially in scenarios where data storage and computational resources are constrained, the ability to selectively retain informative data points becomes paramount. The proposed algorithm leverages the Grad-CAM technique, a visualization method designed to highlight regions in an image that a neural network deems important for its predictions. By quantifying the importance of these regions, We can make informed decisions about which data points to retain for potential retraining and which to discard as redundant or routine.

A key component of the algorithm is the **DRIP Score** (DRIPS), a metric derived from Grad-CAM heatmaps to measure the informativeness of data points. The **DRIPS** quantifies the average activation within the Grad-CAM heatmap for a given input, representing the importance of that data point to the model's prediction. Mathematically, the DRIPS for a data point I is computed as:

$$\text{DRIPS}(I) = \frac{\sum_{i,j} H(I)_{i,j}}{W \times H}, \tag{3.8}$$

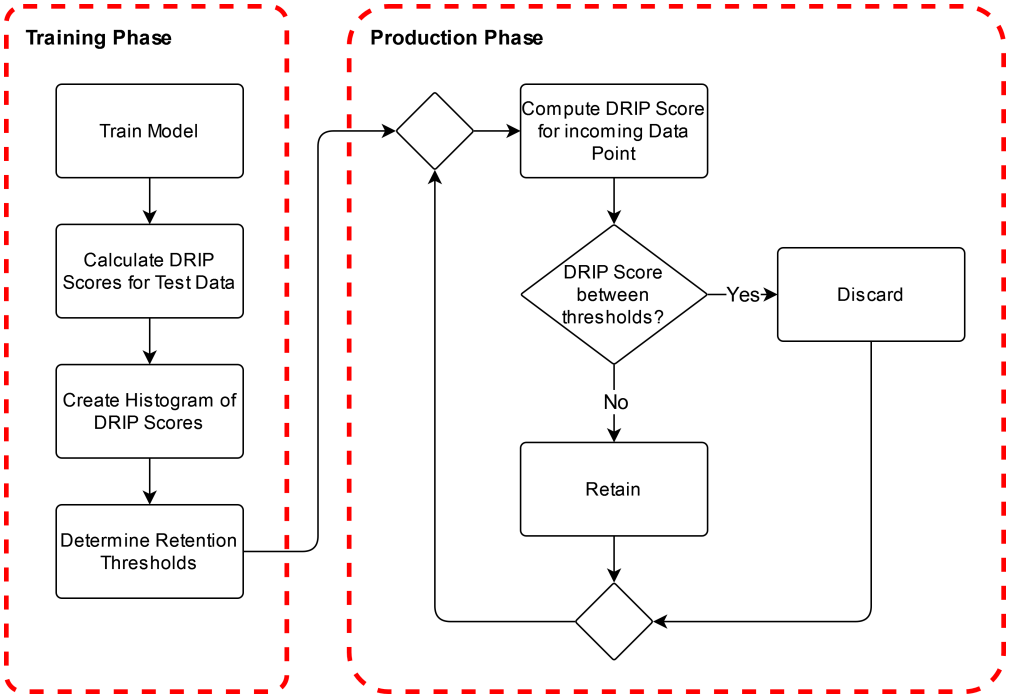

Figure 3.1: Flowchart illustrating the seven-step process of the DRIP algorithm. The flowchart provides a visual representation of the algorithm's sequential steps, from initial model training to the final decision on data point retention in real-time scenarios.

where $H(I)$ is the Grad-CAM heatmap for I, W and H are the width and height of the heatmap, and i, j are pixel indices.

The average of a Grad-CAM heatmap provides a simple yet powerful metric to evaluate the importance of a data point. By aggregating activations across the heatmap, this approach captures the overall influence of the input on the model's decision. Inputs with high average activations indicate strong alignment with the model's learned features, making them critical for retraining or further analysis. Conversely, low averages suggest redundancy or irrelevance, guiding the selective retention of data points. This computationally efficient method aligns well with the constraints of embedded systems, ensuring resource-aware decision-making without compromising model performance.

The algorithm operates in two distinct phases: the *Training Phase* and the *Production Phase*. The Training Phase establishes thresholds based on the distribution of DRIP Scores in a test dataset. These thresholds are then utilized in the Production Phase to evaluate incoming data points in real-time, deciding their retention based on their computed importance.

- **Training Phase:** This phase involves training the neural network model, computing DRIP Scores for a test dataset, and establishing retention thresholds based on the distribution of heatmap values.

- **Production Phase:** In this phase, for each new data point encountered in a production environment, its Grad-CAM heatmap is computed. The data point's retention is then decided based on whether its DRIP Score falls within the thresholds established during the Training Phase.

The subsequent subsections provide a detailed breakdown of each phase, elucidating the steps involved and the rationale behind them. Fig 3.1 provides a graphical overview of the sequence of the DRIP algorithm

3.2.6 Training Phase

The training phase is crucial for establishing the thresholds that will be used in the production phase to determine the importance of a data point. The steps are as follows:

1. **Model Training:** Train a neural network model using the training dataset.
2. **Compute DRIPS (DRIP Score):** For each image I in the test dataset, compute the Grad-CAM heatmap $H(I)$. For each Grad-CAM heatmap $H(I)$, calculate the average value of the heatmap. This is done using the formula 3.8
3. **Histogram Creation:** Construct n histograms (n represents the number of classes in the dataset) using the DRIPS values of all the images in the test dataset. These histograms will show the distribution of DRIPS values across the dataset.
4. **Determine Retention Thresholds**: This step involves identifying consistent regions within the DRIPS score distributions, assumed to contain less informative data points. The process is as follows:
    a) **Sort DRIPS Scores**: For each class, sort the DRIPS scores in ascending order.
    b) **Define Discard Percentage Window (DPW)**: Set a window size as a percentage of the total number of scores.
    c) **Slide Window and Calculate Standard Deviation**: Slide this window across the sorted scores. At each position, calculate the standard deviation of the scores within the window.
    d) **Find Minimum Standard Deviation Window**: Locate the window where the standard deviation is minimal. This window presumably contains the least informative scores. The assumption here is grounded in the nature of the Grad-CAM-based DRIP Scores: a lower standard deviation within a small window indicates that the data points in that range are relatively uniform in their Grad-CAM heatmap activations. Such uniformity is typically observed in data points that contribute minimally to model decision-making, as their activations lack the variability that often signals critical or unique features. By identifying the window with minimal standard deviation, We are isolating data points that likely represent routine or redundant patterns, which are less

informative for retraining purposes. This approach ensures that the algorithm prioritizes data retention for more diverse and informative instances.

e) **Set L_{lower} and L_{upper}**: The thresholds are set based on this window's minimum and maximum scores. The lower threshold L_{lower} is the minimum score in this window:

$$L_{\text{lower}} = \min\left(\text{DRIPS in the Selected Window}\right) \tag{3.9}$$

The upper threshold L_{upper} is the maximum score in the window:

$$L_{\text{upper}} = \max\left(\text{DRIPS in the Selected Window}\right) \tag{3.10}$$

This is shown in Fig. 3.2.

f) **Thresholds for Each Class**: Repeat steps (a)-(e) for each class to calculate class-specific L_{lower} and L_{upper}.

The calculated limits L_{lower} and L_{upper} will serve as thresholds during the production phase to decide whether to retain or discard a data point.

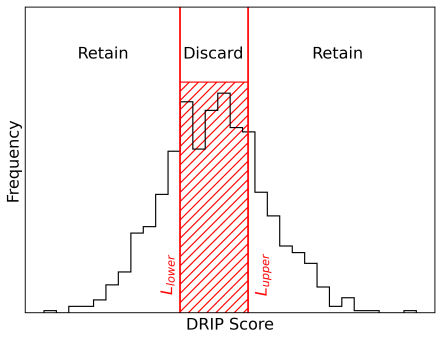

Figure 3.2: Determination of retention thresholds from an exemplary DRIP Scores. The peak represents the highest accumulation of DRIP Scores. The calculated lower (L_{lower}) and upper (L_{upper}) limits encapsulate 25% of the DRIP Scores, serving as the criteria for my algorithm's data retention decisions.

3.2.7 Production Phase

Once the thresholds have been established in the training phase, the production phase uses these thresholds to decide the importance of incoming data points. The steps for this phase are as follows:

1. **Compute DRIPS for New Data Point:** For a new data point (image) I_{new} processed in production, compute its Grad-CAM heatmap $H(I_{\text{new}})$. The Grad-CAM heatmap highlights the regions of the input that have the most influence on the model's prediction. To ensure the heatmap accurately reflects the areas

influencing the model's decision, the predicted label obtained from the model is used. Next, calculate the DRIPS for this heatmap using the formula:

$$\text{DRIPS}(I_{\text{new}}) = \frac{\sum_{i,j} H(I_{\text{new}})_{i,j}}{W \times H} \tag{3.11}$$

where W and H are the width and height of the image I_{new}, respectively, and i, j are pixel indices.

The DRIPS value quantifies the average activation of the Grad-CAM heatmap, which corresponds to the degree of importance or informativeness of the data point for the model's prediction. A higher DRIPS indicates that the data point contributes significantly to the model's decision, while a lower DRIPS suggests that the data point is less impactful.

2. **Decision Making:** Compare the computed DRIPS value against the thresholds determined in the training phase:

$$L_{\text{lower}} \leq \text{DRIPS}(I_{\text{new}}) \leq L_{\text{upper}} \tag{3.12}$$

If the DRIPS value falls within this range, the data point is deemed routine or redundant and is discarded. Otherwise, the data point is considered important or informative and is retained for potential retraining or further analysis.

The reasoning behind this approach is rooted in the assumption that data points with DRIPS values within the threshold range are likely to represent patterns already well-learned by the model. These data points contribute little new information and can be safely discarded without impacting model performance. Conversely, data points with DRIPS values outside the threshold range may represent novel or underrepresented patterns, making them valuable for retraining and improving the model's generalization.

Rationale and Explanation: The underlying idea of this method is to leverage the interpretability of Grad-CAM heatmaps to prioritize data retention based on the informativeness of data points. By focusing on the regions of the input that influence the model's prediction, the DRIP algorithm quantifies how much a given data point contributes to the learning process. Data points with low variability in their Grad-CAM activations (as determined during the training phase) are likely to be redundant, as they align closely with the model's existing knowledge. In contrast, data points with high or outlier DRIPS values suggest new or diverse information that the model has not fully captured.

3.2.8 Computational Efficiency

The computational overhead in the **production phase** is minimal, primarily involving the generation of Grad-CAM heatmaps and the computation of DRIP scores for each new data point. The key computational steps are as follows:

1. Grad-CAM Heatmap Generation Grad-CAM computes feature importance by *backpropagating gradients* of the target class score through the last convolutional layer. Given an input of size $N \times N$, the feature maps in the last convolutional layer typically have a reduced spatial resolution of $M \times M$, where $M \ll N$. The computational complexity of Grad-CAM heatmap generation is:

$$O(B \cdot C \cdot M^2) \tag{3.13}$$

where:

- B is the batch size (typically 1 for real-time inference),
- C is the number of feature maps in the selected layer,
- M^2 is the number of spatial locations in the feature maps.

2. DRIP Score Calculation Once the heatmap is generated, the DRIP Score is computed as:

$$S_{\text{DRIP}}(x) = \frac{1}{M^2} \sum_{i,j} H(x)_{i,j} \tag{3.14}$$

This requires a simple summation and normalization over M^2 pixels, which runs in:

$$O(M^2) \tag{3.15}$$

3. Overall Complexity in Production Phase Since $O(B \cdot C \cdot M^2)$ dominates over $O(M^2)$, the total complexity per data point is:

$$O(B \cdot C \cdot M^2) \tag{3.16}$$

For small feature maps ($M \ll N$), this remains **significantly more efficient than full backpropagation**.

Suitability for Edge Devices

- The computationally intensive steps, such as **model training and DRIP threshold determination**, occur in the **training phase** and can be offloaded to cloud or server environments.
- In the **production phase**, DRIP operates with minimal per-sample overhead, making it feasible for **resource-constrained edge devices**.

3.2.9 Pseudocode

The following pseudocode outlines the main steps of the DRIP algorithm, detailing the process of computing DRIP scores and making data retention decisions based on threshold analysis.

Algorithm 1 Selective Data Retention using DRIP algorithm

Require: TrainDataset: Dataset used for training the model
Require: TestDataset: Dataset used for evaluating the model
Require: Model: Neural network model
Require: THP: Threshold percentage in %. Here 25%
Require: NewDataPoint: New data point encountered in production
Ensure: Decision: Whether to retain the NewDataPoint or not

Train Model using TrainDataset
Initialize empty list: DRIPS_List
for each Image in TestDataset **do**
    Compute Grad-CAM heatmap for Datapoint
    Compute DRIP Score for Datapoint:
    DRIP Score = $\frac{\sum_{i=1}^{W}\sum_{j=1}^{H} H(i,j)}{W \times H}$
    Append DRIP Score to DRIP_List
end for
Create histograms of DRIPS_List
Determine peak of the histograms
Calculate lower and upper thresholds for THP data around the peaks of the histograms
Compute Grad-CAM heatmap for NewDataPoint
Compute DRIPS for NewDataPoint of the corresponding class
if DRIPS is between the lower and upper thresholds **then**
    Decision = "Discard"
else
    Decision = "Retain"
end ifreturn Decision
Fine-tune Model

3.3 Case Studies: Data Selection in Edge ML

This section presents empirical case studies that evaluate the effectiveness of the proposed data selection approach using DRIP. The following experiments investigate its impact on model accuracy, storage efficiency, and robustness across various datasets and application domains.

3.3.1 Experiments

Objective: The primary aim of my experiment is to validate the effectiveness of the DRIP algorithm in enhancing model performance and ensuring efficiency in data storage.

Datasplit

- **Training Dataset 40%:** A labeled dataset utilized for the initial training of the model.
- **Validation Dataset 20%:** A separate labeled set to test the model and generate DRIP Scores.
- **Production Dataset 40%:** Simulated or real-world unlabeled data points that the model will encounter in a production-like scenario.

Experimental Setup

Baseline Model: is designed to establish a foundational level of performance against which other models can be compared. This process begins by training a neural network model using the entire training dataset (Dataset 1). After the model is adequately trained, its performance is evaluated on a separate test dataset (Dataset 2). The accuracy obtained from this evaluation serves as the baseline accuracy, providing a benchmark to assess the improvements made by more advanced models and techniques.

DRIP Model: introduces an advanced method to improve the performance of the neural network model by selectively retaining important data points. Initially, the model is trained using the training dataset (Dataset 1). The proposed DRIPS algorithm is then applied to this dataset to determine the DRIPS thresholds, which help in identifying significant data points. This algorithm is simulated in a production environment to decide which data points from the production dataset should be retained (Retained Data). The model is subsequently retrained using these retained data points, and its performance is evaluated on the separate test dataset (Dataset 2). This approach aims to enhance model accuracy by focusing on the most relevant data.

All-Data Model: aims to maximize the utilization of available data by combining the entire training dataset with the production dataset. In this approach, the neural network

model is first trained using the entire training dataset (Dataset 1) alongside the production dataset (Dataset 3). This comprehensive training process leverages all accessible data to potentially improve the model's learning capabilities. The model's performance is then evaluated on a separate test dataset (Dataset 2) to determine its accuracy. This metric is used to compare against the DRIP Model, providing insights into the effectiveness of using all available data versus selectively retained data points.

Evaluation Metrics

- **Model Performance:** Metrics such as accuracy, F1-score, etc., on the test dataset.
- **Data Retention Rate:** Percentage of data points retained from the production dataset.
- **Computational Efficiency:** Time taken for each data point's data retention decision.
- **Storage Savings:** Amount of storage saved due to selective data retention.

Procedure

In order to generate meaningful results, each experiment was carried out 20 times for each data set. In the results Table 3.1, We showed the average and the standard deviation of the results. Before each run, all data from the use case was randomly assigned to the 3 datasets. Afterward, the following 4 steps are carried out (this process is shown in Fig. 3.3):

1. **Train the Baseline Model:** Train the model using the entire training dataset (Dataset 1) and evaluate its performance on the test dataset (dataset 2).
2. **Apply the DRIP algorithm:** Determine the DRIP Score thresholds using dataset 1 and decide which data points to retain from the production dataset (dataset 3).
3. **Retrain and Evaluate:** Retrain the model using the retained data points and evaluate its performance on the test dataset (dataset 2).
4. **Comparison:** Analyse the performance of the retrained model against the baseline and the accuracy with retraining with all data, considering data retention rate, computational efficiency, and storage efficiency.

3.3.2 Overview of Evaluated Datasets

To evaluate the efficacy of my algorithm across diverse domains, We tested it on four distinct datasets, each representing different areas of application. Detailed descriptions of these datasets, including their structure and specific characteristics, can be found in Appendix A. Below is an overview of the datasets referenced in this study:

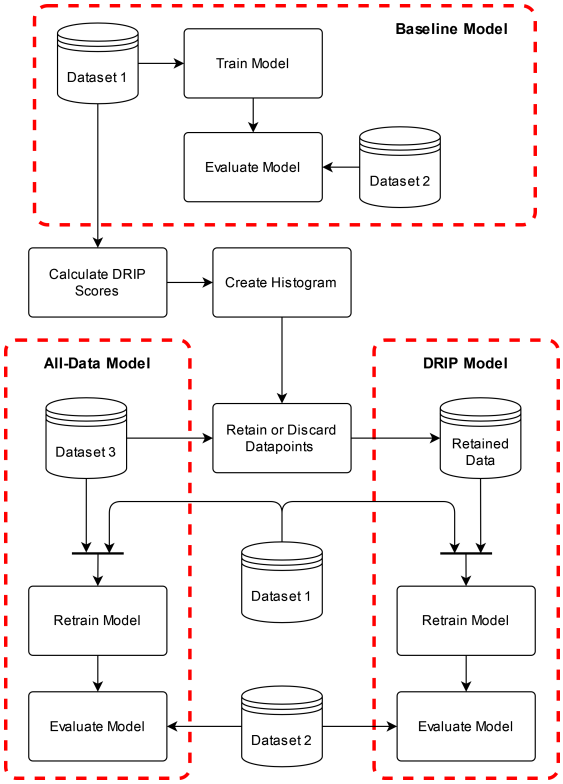

Figure 3.3: Schematic representation of the experimental process detailing the computation of the three key metrics: Baseline Model Accuracy, All-Data Model Accuracy, and DRIP Model Accuracy

1. **MNIST:** A collection of grayscale images of handwritten digits. This dataset is widely used for evaluating image classification models. For more details, refer to Appendix A.0.1.

2. **CIFAR-10:** A dataset of 60,000 color images across 10 classes, commonly used in computer vision tasks. Further information can be found in Appendix A.0.2.

3. **Plant Disease (PD):** A dataset comprising RGB images of healthy and diseased crop leaves, categorized into 38 classes. See Appendix A.0.3 for additional details.

4. **Speech Commands (SC):** An audio dataset containing one-second-long clips of spoken words, designed for keyword spotting tasks. A detailed description is provided in Appendix A.0.4.

3.3.3 Analysis of Outcomes

My evaluation of the DRIP algorithm across four benchmark datasets demonstrates its efficacy in selective data retention, optimizing storage, and maintaining or enhancing model accuracy. This section presents my findings, focusing on storage savings, the impact of varying DPW size, and the algorithm's robustness to noise.

Table 3.1: Summary of results for MNIST, CIFAR-10, Plant Disease (PD) and Speech Commands (SC) datasets. The mean and standard deviation of the 20 runs is calculated.

Dataset	**Baseline Acc.**	**All-Data Acc.**	**DRIP Acc.**	**Random Acc.**	**Storage Savings**
MNIST	97.8% (±0.2%)	98.9% (±0.1%)	98.9% (±0.1%)	98.2% (±0.5%)	39% (±1%)
CIFAR-10	87.5% (±0.3%)	89.1% (±0.2%)	89.2% (±0.2%)	88.8% (±0.6%)	23% (±2%)
PD	71.0% (±0.2%)	77.5% (±0.2%)	77.4% (±0.2%)	73.7% (±0.5%)	35% (±1%)
SC	76.6% (±0.4%)	85.6% (±0.5%)	85.7% (±0.4%)	80.1% (±0.7%)	29% (±3%)

DRIP's Impact on Storage Efficiency

The DRIP algorithm significantly improved storage efficiency across all tested datasets by selectively retaining only the most informative data points, as shown in Table 3.1. This approach not only preserved but sometimes enhanced model accuracy compared to using all available data. In the context of my experiments, accuracy refers to the percentage of correct predictions made by the model on a test dataset. It is measured by comparing the model's output against the true labels of the data points. A higher accuracy indicates that the model is performing better at correctly classifying or recognizing patterns in the dataset.

Storage savings, on the other hand, represents the percentage reduction in the amount of data stored during the model's training or retraining process. By selectively retaining only the most informative data points, the DRIP algorithm reduces the total amount of data that needs to be stored on the device, thereby saving storage space. This is particularly important for on-device training in resource-constrained environments, where memory is limited. Here is DRIP's impact on each dataset:

CIFAR-10: Achieved 89.2% mean accuracy, slightly higher than the all-data model's 89.1%, with a 23% mean reduction in storage. [76]

MNIST: Matched the all-data model's 98.9% mean accuracy with a 39% mean storage saving, demonstrating effectiveness in high-accuracy datasets. [106]

Speech Commands: Slightly outperformed the all-data model (85.7% vs. 85.6%) with a 29% mean storage reduction, indicating adaptability to audio data. [77]

Plant Disease: Closely matched the all-data model's 77.4% mean accuracy with a 35% mean reduction in storage, highlighting potential in storage-constrained applications. [107]

These results illustrate DRIP's capability to maintain or enhance model accuracy across diverse datasets while significantly reducing storage requirements, making it valuable for on-device applications with limited storage and computational resources.

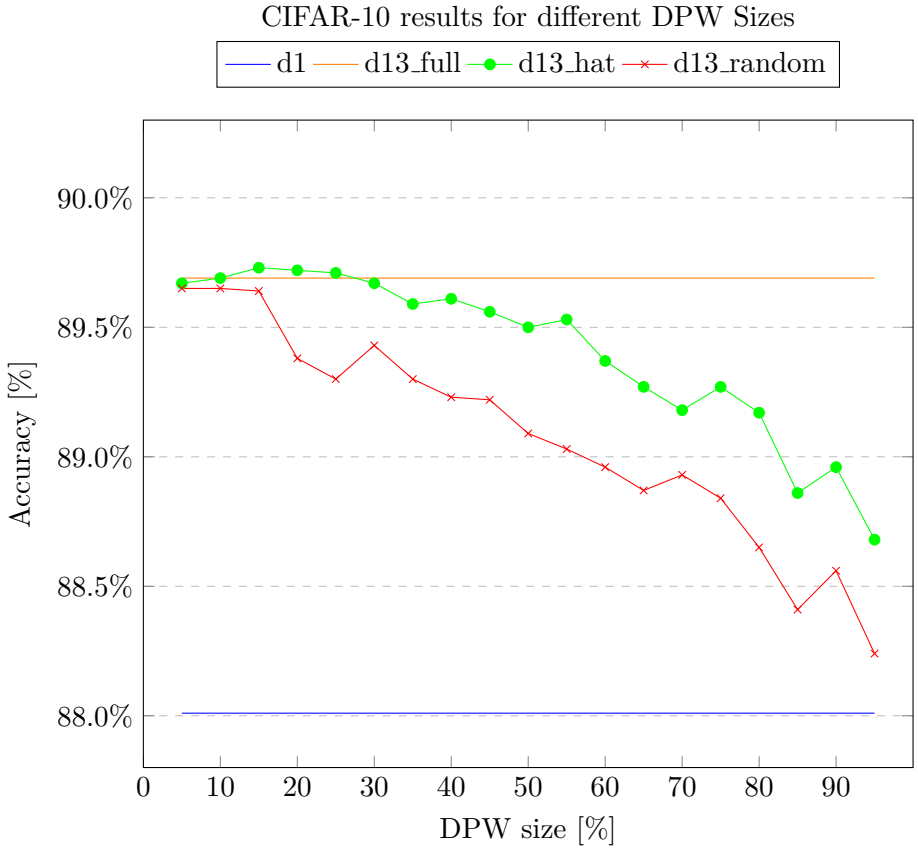

Figure 3.4: Analysis of CIFAR-10 Model Accuracy Across Various DPW sizes: This graph compares the accuracy of different configurations (d1, d13_full, d13_hat, d13_random) as the DPW size changes, highlighting the algorithm's sensitivity to parameter adjustments and its impact on model accuracy.

Investigation of Different DPW Sizes

My analysis extends to exploring different Discard Percentage Window (DPW) sizes, examining their impact on model accuracy. The CIFAR-10 dataset, for example, showed in Figure 3.4 a slight improvement in accuracy with DRIP over the all-data model, highlighting the algorithm's adeptness at handling varying levels of data retention. Figure 3.5 illustrates the model accuracy across different DPW sizes, showcasing DRIP's consistent

performance even as the bandwidth adjustments are made.

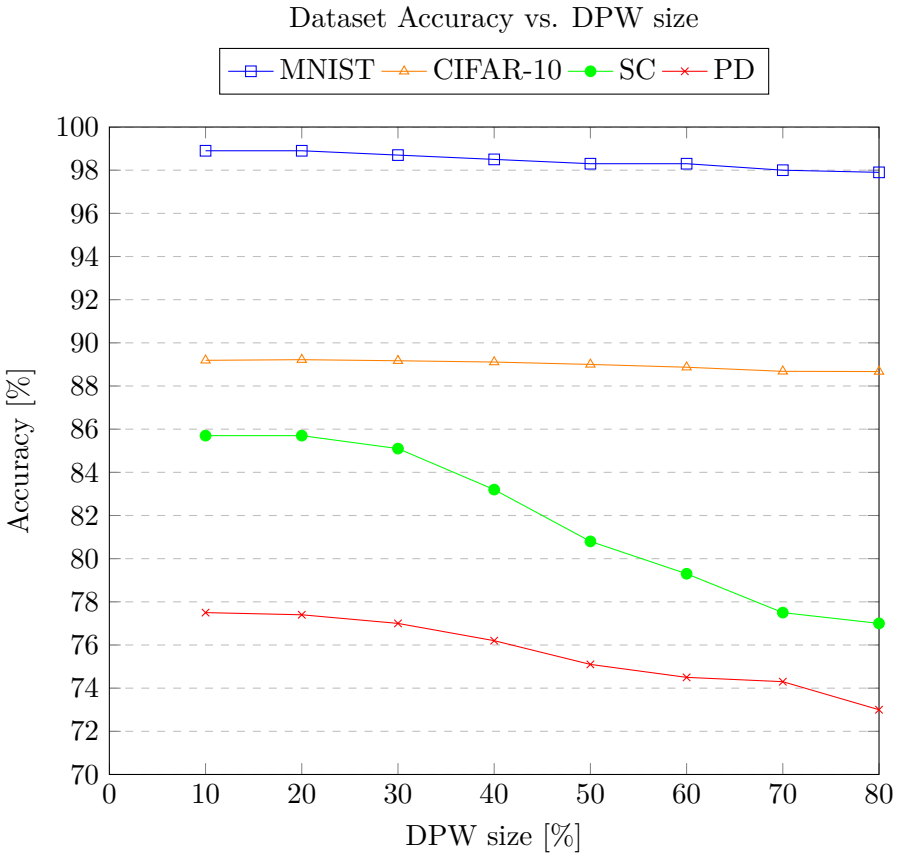

Figure 3.5: Model accuracy across datasets with varying DPW sizes: Demonstrates the effect of different DPW sizes on the accuracy of the datasets.

Robustness to Noise

To evaluate the robustness of DRIP (DRop unImportant data Points) against noisy data, We conducted experiments on the CIFAR-10 dataset by introducing noise and mislabeling. We incrementally added noise in steps of 10% and retrained the model twice at each noise level: once with the unfiltered dataset and once with the dataset filtered by DRIP.

The results, shown in Figure 3.6, illustrate that as noise increases, the accuracy of the model trained on the unfiltered dataset declines more sharply compared to the model trained with data filtered by DRIP. These findings demonstrate DRIP's effectiveness in handling noisy data by selectively retaining more reliable and informative data points, thus enhancing overall model performance and resilience.

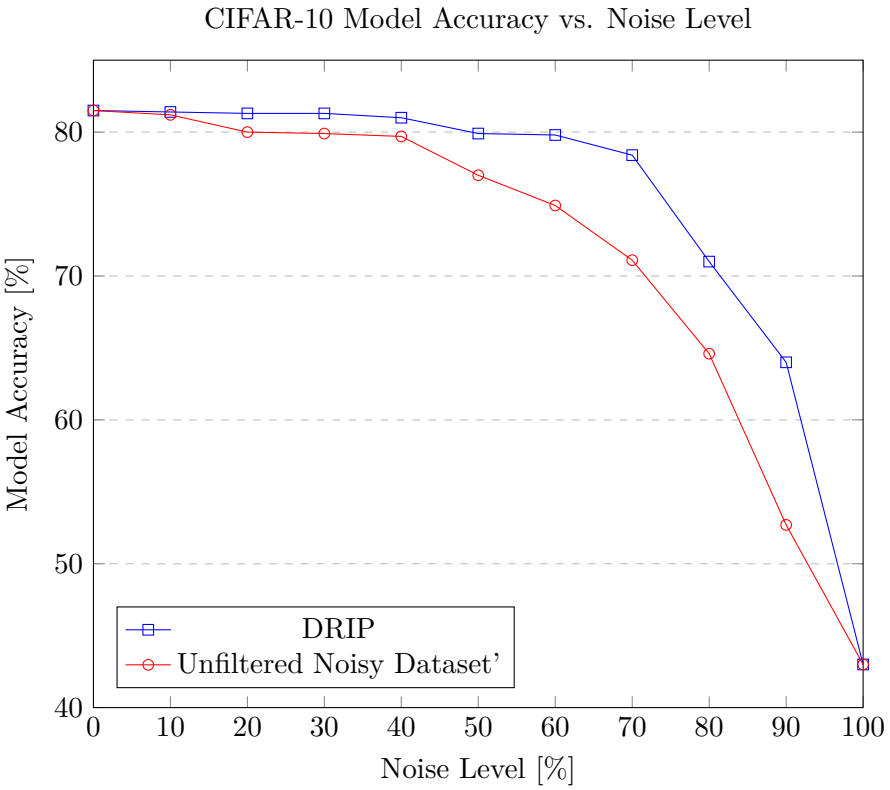

Figure 3.6: Model Accuracy Comparison under Increasing Noise Levels - Demonstrating DRIP's superior performance in maintaining accuracy against unfiltered noisy data in CIFAR-10.

3.4 Discussion on Efficiency and Performance Observations

The results presented in Table 3.1 offer valuable insights into the efficency of the proposed selective data retention algorithm using the DRIP algorithm.

The evaluation of the DRIP algorithm across diverse datasets highlights its effectiveness in selective data retention and storage optimization, while maintaining or even enhancing model accuracy. This discussion elaborates on the key findings and provides a deeper analysis of the implications, supported by specific evidence from the results.

Comparison with Existing Data Selection Techniques

The DRIP algorithm, while novel and efficient, can be contextualized against several established data selection techniques to highlight its advantages and unique contributions. This discussion provides a comparison with methods such as random selection, active learning, coreset selection, and Grad-CAM-based approaches.

Random Selection: Random data selection is one of the simplest methods where data points are retained or discarded randomly without considering their individual importance. While this approach is computationally efficient, it lacks the ability to prioritize informative or diverse data points. This can lead to the retention of redundant or irrelevant data, resulting in suboptimal model performance. As demonstrated in Table 3.1, the DRIP algorithm outperforms random selection across all datasets, achieving higher accuracy while retaining a smaller percentage of the data. For example, in the MNIST dataset, DRIP achieved an accuracy of 98.9% with 39% storage savings, compared to random selection's accuracy of 98.2%. This improvement underscores the effectiveness of DRIPS informed retention strategy over random selection.

Active Learning: Active learning focuses on selectively labeling the most informative data points for training, typically using criteria like uncertainty sampling or diversity maximization. While active learning effectively reduces the labeling cost and enhances model performance, it assumes the availability of an oracle (e.g., a human annotator) to label selected data points, which may not be feasible in real-time or on-device scenarios. In contrast, the DRIP algorithm operates autonomously by leveraging Grad-CAM heatmaps to evaluate the importance of data points. This independence from external labeling resources makes DRIP particularly suitable for on-device applications in resource-constrained environments.

Coreset Selection: Coreset selection techniques aim to construct a smaller representative subset of the original dataset that retains the essential statistical and geometric properties of the data. While methods like K-Center and greedy algorithms are effective in reducing data size, they often require significant computational resources to evaluate pairwise distances or optimize subset selection. In comparison, DRIPS computational complexity is significantly lower due to its reliance on Grad-CAM heatmaps and a simple scoring mechanism. Additionally, DRIPS focus on real-time data retention enables dynamic adaptation to evolving data distributions, a feature often lacking in static coreset approaches.

Advantages of the DRIP Algorithm: The key advantages of the DRIP algorithm over existing techniques include:

- **Computational Efficiency for Real-Time and Resource-Constrained Environments:** As discussed in Section 3.2.8, DRIP operates with minimal computational overhead, making it well-suited for real-time applications and resource-constrained environments. By leveraging Grad-CAM heatmaps and a lightweight scoring mechanism, DRIP avoids expensive pairwise distance computations or iterative optimizations, ensuring efficient execution on edge devices.
- **Storage Efficiency:** DRIP achieves significant storage savings while maintaining or enhancing model accuracy, as demonstrated across diverse datasets in Table 3.1.

- **Independence from External Resources:** Unlike active learning, DRIP does not rely on external oracles for labeling, enabling fully autonomous operation.

3.4.1 Model Performance and Efficiency

The DRIP algorithm demonstrated strong performance across all tested datasets, including CIFAR-10, MNIST, Speech Commands, and Plant Disease. In terms of accuracy, the algorithm consistently matched or exceeded models trained on the entire dataset. For instance, in CIFAR-10, DRIP achieved a higher accuracy (89.2%) than the all-data model (89.1%), while for MNIST, it achieved the same 98.9% accuracy as the all-data model. This suggests that selectively retaining data based on the DRIP Score does not compromise, and may even improve, model performance.

The slight improvement in accuracy observed in CIFAR-10 and Speech Commands suggests that DRIPS selective retention of the most informative data points may improve the model's ability to generalize. By training on a refined subset of data that contributes meaningfully to learning, the algorithm prevents overfitting and ensures the model is exposed to data points that drive learning outcomes. This aligns with the general observation that data quality often trumps quantity in machine learning.

3.4.2 Efficiency in Data Retention and Storage Savings

One of the key benefits of the DRIP algorithm is its ability to achieve significant storage savings without compromising performance. Across the four datasets, DRIP selectively retained between 61% and 77% of the original data, resulting in storage savings ranging from 23% to 39%. For instance, the MNIST dataset showed a 39% reduction in stored data while maintaining the same accuracy as the all-data model. This level of efficiency is especially valuable in resource-constrained environments such as edge computing or IoT devices, where storage is at a premium.

The storage savings do not come at the cost of learning efficacy. In fact, by filtering out redundant or less informative data points, DRIP ensures that the model is retrained on high-quality data, which contributes directly to model performance. This is particularly advantageous for on-device applications, where not only storage but also computational power is limited.

3.4.3 Adaptability and Real-Time Decision Making

A notable strength of DRIP is its adaptability to a variety of data types. Whether applied to image datasets like CIFAR-10 and MNIST or audio data like Speech Commands, DRIP consistently performs well. This versatility stems from its reliance on Grad-CAM, which is a general-purpose interpretability method applicable across different neural network architectures and modalities. Since Grad-CAM derives feature importance directly from the model's learned representations, it naturally extends to different data domains where convolutional structures are applicable, such as vision, audio, and even structured sensor data.

Justification for Adaptability To formally justify DRIP's adaptability, consider a trained neural network $f(\cdot)$ mapping an input space X to a class prediction space Y. The relevance of an input sample $x \in X$ to the model's decision can be quantified through its Grad-CAM heatmap $H(x)$. The DRIP Score is then computed as:

$$S_{\text{DRIP}}(x) = \frac{1}{Z} \sum_{i,j} H(x)_{i,j} \tag{3.17}$$

where Z is the total number of elements in $H(x)$, ensuring a normalized importance measure.

Adaptability Theorem: If Grad-CAM provides meaningful explanations for model predictions across different modalities, then the DRIP Score remains a valid measure of data importance across these modalities.

Derivation The adaptability of DRIP across different data modalities can be justified as follows:

1. Grad-CAM computes feature importance by backpropagating class-specific gradients to a target convolutional layer. This process is architecture-agnostic and depends only on the existence of a differentiable feature extraction layer.

2. Since modern deep learning models in both vision (CNNs) and audio (1D-CNNs, spectrogram-based CNNs) use convolutional structures, Grad-CAM can generate meaningful importance heatmaps for both domains.

3. The DRIP Score is a direct aggregation of these importance values, meaning its computation remains valid across different data modalities as long as Grad-CAM highlights relevant features.

4. Empirical validation (as shown in Table 3.1) confirms that DRIP consistently selects useful data points regardless of the dataset type, reinforcing its adaptability.

Thus, DRIP inherits Grad-CAM's broad applicability, making it suitable for diverse machine learning tasks beyond image classification.

Justification for Real-time Decision Making As discussed in Section 3.2.8, DRIP's ability to make real-time decisions about data retention is critical for applications requiring immediate processing, such as autonomous systems and mobile devices. The efficiency of the DRIP Score computation allows rapid evaluation of incoming data points, ensuring a low-latency decision-making process.

3.4.4 General Observations

Several general observations from the experimental results highlight the broader advantages of the DRIP algorithm:

- **Consistent Performance Across Datasets:** DRIP's performance across diverse datasets underscores the robustness of the approach, making it applicable to a wide range of machine learning tasks and environments.
- **Quality over Quantity:** The consistent accuracy of models trained with selectively retained data points illustrates the importance of focusing on high-quality, informative data rather than large volumes of data.
- **Real-time Efficiency:** The ability to make data retention decisions in real time offers practical advantages for on-device machine learning, where both storage and response time are critical.

3.4.5 Limitations and Future Directions

While DRIP offers substantial advantages, there are limitations to consider. The initial training phase, where thresholds for DRIP scores are determined, can be computationally intensive, particularly when working with large datasets. This phase could benefit from optimization, such as incorporating parallel processing or offloading it to cloud environments.

Additionally, the algorithm's effectiveness depends on the quality of the initial dataset. If the training data used to calculate DRIP scores is not representative of the overall data distribution, the algorithm may retain uninformative or biased data points, leading to suboptimal performance. Future research could explore methods to dynamically adjust the DRIP score thresholds or integrate other visualization techniques to further enhance Data Selection accuracy.

Furthermore, real-world deployments of DRIP on edge devices would provide valuable insights into its practical applications, particularly in dynamic and evolving environments where data characteristics change over time.

3.4.6 Broader Impacts and Ethical Considerations

DRIPS efficient data retention approach has the potential to significantly reduce storage and computational costs, making advanced machine learning technologies more accessible and sustainable, particularly in resource-constrained environments. However, the selective retention process may introduce biases if the initial training data is not representative. This could result in models that perform well for certain data subgroups but poorly for others. Ensuring that the training datasets are diverse and representative is crucial to mitigating this risk. In addition, regular monitoring of algorithmic performance across various demographics and data sources is essential to ensure fairness and avoid unintended biases in decision-making.

Chapter 4

Novel Method to Enhance Memory Efficiency in Edge ML with Advanced Incremental Learning Techniques

Incremental learning offers a promising avenue for updating models on-device without the need for extensive retraining [17]. However, traditional methods often fall short in terms of memory efficiency—a critical component for Edge ML applications. This chapter explores advanced techniques in incremental learning tailored to improve memory utilization without compromising the learning efficacy on embedded devices.

4.1 Fundamentals of Incremental Learning

Incremental learning is a critical aspect of machine learning, particularly relevant in the context of Edge ML, where models must adapt to new data continuously without the need for complete retraining from scratch. This approach is essential for efficient learning and adaptation in environments where data evolves over time or where it is impractical to store and retrain on the entire dataset due to resource constraints.

4.1.1 Principles of Incremental Learning

Incremental learning in machine learning focuses on the model's ability to learn from a stream of data over time. Mathematically, this involves updating the model's parameters incrementally as new data arrives, ensuring that the model remains relevant and accurate without the need to retrain from scratch on the entire dataset [93].

A fundamental approach to incremental learning is through the concept of online learning, where the model updates its parameters θ after observing each new data point or a mini-batch of data points. Given a loss function $L(\theta, x_t, y_t)$ that measures the prediction error on the new data point (x_t, y_t) at time t, the model parameters θ are updated using gradient descent as follows:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t, x_t, y_t) \tag{4.1}$$

where η is the learning rate, and $\nabla_\theta L(\theta_t, x_t, y_t)$ is the gradient of the loss with respect to the parameters θ at time t.

To address the challenge of catastrophic forgetting, where a model tends to forget previously learned information upon learning new data, regularization techniques such as Elastic Weight Consolidation (EWC) [78] are used. EWC adds a regularization term to the loss function that penalizes changes to parameters important for previous tasks. The modified loss function becomes:

$$L_{EWC}(\theta) = L(\theta, x_t, y_t) + \lambda \sum_i F_i(\theta_i - \theta_{i,old})^2 \tag{4.2}$$

where λ is a regularization coefficient, F_i is the Fisher Information Matrix that measures the importance of parameter θ_i for previous tasks, and $\theta_{i,old}$ are the parameter values before learning the new data. This term encourages the model to keep critical parameters close to their old values, mitigating forgetting.

Incremental learning's mathematical formulations highlight its dynamic and adaptive nature, making it particularly suitable for applications in Edge ML where models must continuously adapt to new data under strict resource limitations.

4.1.2 Relevance in Edge ML

The integration of incremental learning within Edge Machine Learning (Edge ML) frameworks is not merely a technical necessity but a strategic imperative to navigate the constraints and capitalize on the opportunities presented by edge computing environments. This chapter elucidates the multifaceted relevance of incremental learning in Edge ML, addressing both its operational benefits and its profound impact on the sustainability and adaptability of edge devices.

- **Addressing Resource Constraints:** At the core of Edge ML's challenges is the limitation of computational and memory resources. Incremental learning, with its capacity to assimilate new information without revisiting the entire dataset, presents a viable solution to this predicament. It allows for efficient use of limited resources by focusing computational efforts on new or unlearned data, thus optimizing the learning process under resource scarcity.
- **Enhancing Model Adaptability:** The dynamic nature of real-world environments necessitates models that can evolve in response to new data or changing conditions. Incremental learning endows Edge ML devices with the ability to adapt over time, ensuring that models remain relevant and accurate. This adaptability is critical in applications where environmental conditions or input data patterns can shift unpredictably.
- **Operational Efficiency and Cost Reduction:** Deploying and maintaining machine learning models in edge devices often entail significant operational and financial costs. Incremental learning reduces these costs by minimizing the need for data transmission back to centralized servers for retraining, thereby also reducing latency and enhancing the responsiveness of Edge ML applications.

- **Sustainability and Lifecycle Extension:** The environmental impact of electronic devices is an increasing concern. By prolonging the operational lifespan of Edge ML devices through adaptive learning capabilities, incremental learning contributes to sustainability. This approach not only extends the useful life of devices but also mitigates the ecological footprint associated with manufacturing and disposing of electronics.

4.2 Efficient Incremental Learning through Dataset Distillation

Incremental learning is a crucial strategy for on-device training, as it enables models to adapt to new data over time without requiring full retraining. This is especially important for resource-constrained edge devices, where storing and processing large datasets is infeasible. However, naive incremental learning can suffer from catastrophic forgetting, where previously learned knowledge is overwritten. To mitigate this, dataset distillation emerges as a powerful approach. By condensing essential knowledge from past data into a compact representation, dataset distillation allows efficient memory utilization while maintaining performance across incremental learning steps. This method ensures that models can continuously learn and adapt in dynamic environments with minimal storage and computational overhead 2.3.

4.2.1 Problem Statement

In this section, We present our general pipeline for all the experiments. We consider a continuous learning or incremental learning scenario in the following sense: Consider a discrete number of N timestamps. At timestamp t, only the current dataset $\mathcal{X}_t$ is available and can be used for training of a neural network. The accuracy of the current and all previously seen data (which is not available) is equally essential. Assume the existence of a test set $\mathcal{T}_t$ corresponding to the data of each step t. After n steps, the best possible accuracy on the entire test $\{\mathcal{T}_t\}_{t=1}^{n}$ and small computational cost and memory are of interest. We assume the network to run on resource-constrained devices with small capability. In order to record the increase of performance over time, We evaluate the model at each step on the entire test set $\{\mathcal{X}_t\}_{t=1}^{n}$.

4.2.2 Incremental Learning in Resource-Constrained Environments

The aforementioned problem of limited memory and computational resources on embedded devices has posed significant challenges in the research and development of on-device training. Our solution to this problem is to use compressed datasets and smaller neural network architectures specifically designed for embedded devices. In this context, We present a multi-step method to address this challenge. The process is depicted in Fig. 4.1 and is described in detail below.

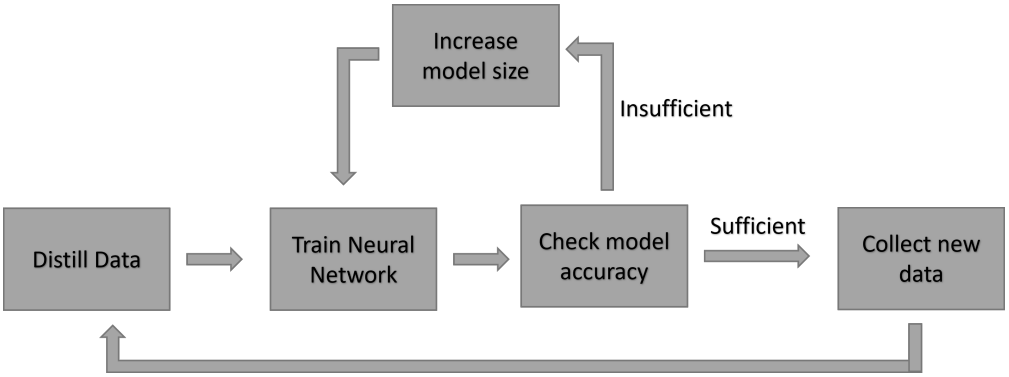

Figure 4.1: The presented method to solve the problem consists of the following steps: 1. Compression of existing data 2. Training of the neural network 3.Adjusting the size of the network as needed. 4. Deployment of the trained network and collection of new data 5. Adding new data to improve performance

1. **Data Distillation:** The foremost step involves distilling the existing data. This process entails carefully selecting the most crucial features and data points relevant to our application, ultimately creating a condensed dataset that requires significantly less storage space.

2. **Neural Network Training:** Post data distillation, We utilize this compressed dataset to train our neural network. The reduced dataset size not only accelerates the training process but also conserves computational resources, making it an efficient approach for embedded devices.

3. **Model Accuracy Validation:** Following the training phase, We evaluate our model's accuracy. This step ensures that our model produces outputs that align as closely as possible to the desired results.

4. **Model Enlargement (if necessary):** If the model's accuracy does not meet the predefined standards—i.e., it performs poorly—we then move to an iterative process of model enlargement. This entails increasing the complexity of our neural network and revisiting Step 2 to re-train the more sophisticated model.

5. **Data Recording and Iteration:** If the model exhibits satisfactory performance, We record the next dataset, and the aforementioned process commences anew. This iterative system ensures our model's consistent growth and improvement, adapting to new data and conditions over time.

The procedure encapsulates a balance between resource utilization and model performance, making it a potentially effective solution for continuous learning scenarios on resource-constrained devices.

4.2.3 Preventing Catastrophic Forgetting

A significant challenge in incremental learning is catastrophic forgetting, where models lose the ability to perform well on previously learned tasks when trained on new data. Our approach mitigates this issue through the use of **data distillation** and **adaptive model enlargement**, which ensure that critical information from past data is retained without overburdening memory or computational resources.

The key principles behind our method are as follows:

- **Distilled Data as Memory Surrogates:** Instead of retaining the entire dataset from previous tasks, We use distilled datasets to encapsulate the essential information required for effective learning. Distillation creates a compact representation of past data while preserving the feature distributions needed to maintain model accuracy. This enables the model to "remember" past tasks without direct access to the original data.
- **Continuous Validation Across Tasks:** At each step, We validate the model on all test sets $\{\mathcal{T}_t\}_{t=1}^{n}$, ensuring that it retains performance on previously seen data. If performance degradation is observed, the distillation process compensates by reinforcing the missing information during training.
- **Adaptive Model Complexity:** Unlike static models, our approach adjusts the network's architecture as new data is encountered. This dynamic resizing allows the model to accommodate increasing task complexity over time while still retaining earlier knowledge, an aspect that traditional fixed-architecture models cannot address.
- **Avoiding Overfitting to New Data:** By mixing distilled data from earlier tasks with the current data during training, We maintain a balanced representation of past and present tasks. This avoids the model overly favoring the most recent data, a common issue in standard incremental learning procedures.

Differences from Existing Methods: Unlike Elastic Weight Consolidation (EWC) or Replay-based methods:

- Our use of distilled datasets significantly reduces memory overhead compared to replaying full datasets.
- We avoid complex parameter-importance calculations like those used in EWC by directly leveraging distilled features that preserve task-relevant information.
- Adaptive model enlargement dynamically scales the model's capacity, unlike most methods that operate with a fixed architecture throughout training.

These principles enable our method to retain knowledge from earlier tasks effectively while adapting to new ones, all within the constraints of resource-limited environments.

4.2.4 Distill Data

Data distillation is a very active research area. Even though our results and accuracies highly depend on the data distillation method, the overall workflow can be applied to almost all of them. In this paper, We use the distribution matching method for images by [108] for illustration, which will be shortly explained in the following. We start by initializing a certain number of images per class (IPC) (e.g. 1 or 10) which We consider as the distilled data. Now, for K steps, We randomly initialize a new convolutional model and measure the difference from the feature space in the last convolutional layer in the network of real data to the feature space of the distilled images. More precisely, We calculate the loss

$$\mathcal{L} = \sum_{c=1}^{C} || \frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}[c]} \psi(x) - \frac{1}{|\mathcal{X}'|} \sum_{x\prime \in \mathcal{X}'[c]} \psi(x') || \tag{4.3}$$

where, C is the number of classes, $\mathcal{X}$ and $\mathcal{X}'$ is the real- and distilled datasets respectively, $(\cdot)[c]$ the subset only containing class c and ψ is map into a feature space depending on the neural network. Finally, We apply a gradient descent update w.r.t. the distilled data, i.e. $\mathcal{X}'_{i+1} = \mathcal{X}'_i - \eta \nabla_{\mathcal{X}'_i} \mathcal{L}_\rangle$ in step $i+1$, where η is a learning rate. Consequently, $\mathcal{X}'$ converges to a similar feature space as the real data, which should result in a similar training performance. We also apply Differentiable Siamese Augmentation (DSA) (omitted in the above notation) which is an augmentation technique suitable for backpropagation ([108]). For a more detailed analysis, see [108].

Compression not only saves storage space but also reduces the computational resources needed to train the neural network.

4.2.5 Train Neural Network

Following the data distillation process, the next stage involves training the neural network using the condensed dataset. This step is critical, as it enables the network to learn and adapt to the distilled data, aligning its output with the target results. By leveraging the distilled dataset, which is significantly smaller than the original data, the training process is accelerated and requires fewer computational resources, making it particularly suitable for embedded devices with limited capabilities.

The training process is iterative and relies on minimizing a cost function, which quantifies the difference between the predicted output and the true output. The cost function, denoted as $\mathcal{C}$, is computed as an average over the cost contributions $\mathcal{C}_x$ of individual training samples:

$$\mathcal{C} = \frac{1}{n} \sum_{x} \mathcal{C}_x \tag{4.4}$$

The optimization process involves adjusting the network's weights and biases to minimize $\mathcal{C}$. This is achieved using gradient descent, where the parameters are iteratively

updated in the direction of the negative gradient of the cost function. The updates for the weights w_k and biases b_k at the k-th iteration are given by:

$$w_{k+1} = w_k - \eta \nabla_w \mathcal{C} \tag{4.5}$$

$$b_{k+1} = b_k - \eta \nabla_b \mathcal{C} \tag{4.6}$$

Here:

- η represents the learning rate, which determines the step size in the optimization process,
- $\nabla_w \mathcal{C}$ and $\nabla_b \mathcal{C}$ are the gradients of the cost function with respect to the weights and biases.

This iterative process, commonly referred to as backpropagation, ensures that the network progressively improves its predictions. For a detailed explanation of forward and backward propagation, refer to Appendix B.1.

By using the distilled dataset $\{\mathcal{X}'_t\}_{i=1}^{t}$, the network achieves a balance between training efficiency and performance. This approach eliminates the need for the network to access the full original dataset, significantly reducing memory and computational requirements. Such efficiency makes this method highly advantageous for resource-constrained environments, such as those encountered in Edge Machine Learning scenarios.

4.2.6 Check Model Accuracy

To evaluate the performance of our model at each incremental learning step, We calculate its accuracy on the unseen validation set $\{\mathcal{T}_t\}_{t=1}^{n}$. This ensures that the model generalizes well to new data and prevents overfitting. Detailed definitions and formulations of accuracy and other evaluation metrics are provided in Appendix B.3.

In our approach, at every step t, We assess the model's accuracy across the entire test set $\{\mathcal{T}_t\}_{t=1}^{n}$ to track its performance on both current and previously seen data. This ensures that the model adapts to new data without sacrificing its ability to perform well on older tasks. The evaluation provides insights into the model's capacity for incremental learning under the constraints of limited resources.

4.2.7 Increase Model Size

The model enlargement stage is an optional phase initiated when the model's accuracy doesn't meet predefined standards or benchmarks—meaning the model is underfitting the data. Underfitting signifies that the model fails to capture the underlying patterns in the data adequately, thereby delivering poor performance during the accuracy validation step.

Model enlargement involves adding complexity to the neural network architecture, such as increasing the number of layers, the number of neurons per layer, or changing the types of layers (e.g., adding convolutional layers for image tasks or LSTM layers for

sequential data). This process is conducted in a structured and controlled manner to ensure effective utilization of resources while addressing the observed performance gaps.

The steps for model enlargement are as follows:

1. **Identify the Bottleneck:** After evaluating the model's performance, We analyze the network layers to identify potential bottlenecks where the model's capacity is insufficient. This could be indicated by consistently high training loss or poor validation accuracy.

2. **Determine the Enlargement Strategy:** Based on the analysis, We select an appropriate strategy for increasing the model's capacity. Examples of such strategies include:
   - **Adding Layers:** Insert additional layers to increase the depth of the network, enabling it to learn more complex hierarchical features.
   - **Increasing Neurons per Layer:** Expand the number of neurons in specific layers, particularly in fully connected layers, to increase the model's representational capacity.
   - **Layer Type Modification:** Replace or augment existing layers with more suitable types for the data, such as using convolutional layers for spatial data or attention mechanisms for sequential data.

3. **Adjust the Architecture:** The model's architecture $\mathcal{M}$ is updated to include the chosen modifications. This adjustment can be mathematically represented as:

$$\mathcal{M}' = \begin{cases} \mathcal{M} + \Delta\mathcal{M} & \text{if } A < A_{\text{standard}} \\ \mathcal{M} & \text{otherwise} \end{cases} \tag{4.7}$$

   Here, $\Delta\mathcal{M}$ refers to the specific modifications applied to the architecture, such as adding layers or increasing neuron counts.

4. **Reinitialize and Retrain:** The modified model $\mathcal{M}'$ is reinitialized and retrained using the distilled data $\{\mathcal{X}'_t\}_{i=1}^{t}$. The retraining focuses on fine-tuning the additional components while preserving knowledge from previous tasks.

5. **Validation and Iteration:** After retraining, the updated model $\mathcal{M}'$ is validated using the dataset $\{\mathcal{T}_t\}_{t=1}^{n}$. If the accuracy still does not meet the predefined standard A_{standard}, further modifications may be applied, repeating the process until satisfactory performance is achieved.

Implementation Details: During enlargement, techniques such as layer freezing and transfer learning are employed to retain knowledge from prior tasks:

- **Layer Freezing:** Existing layers are frozen (i.e., their weights are not updated) to prevent overwriting previously learned features.
- **Transfer Learning:** Pretrained weights from similar tasks can be utilized to initialize the new layers, accelerating convergence and improving performance.

Avoiding Overfitting: To prevent overfitting due to increased model capacity:

- Regularization techniques, such as dropout and weight decay, are applied to the enlarged components.
- Validation performance is monitored closely, with early stopping implemented to halt training if overfitting is detected.

By adaptively adjusting the complexity of the model through these structured steps, this method ensures that the model remains appropriate for the data complexity at each stage of incremental learning. This approach provides an efficient solution for resource-constrained devices, facilitating performance optimization without unnecessary resource consumption.

4.2.8 Data Recording and Iteration

The final step in this method is data recording and iteration. After the model has been trained and validated, and possibly enlarged, the process moves forward with the next dataset. This step involves recording or acquiring new data, which is then subjected to the complete sequence of steps described earlier.

Suppose $\mathcal{X}_{t+1}$ represents the new dataset obtained at the $(t+1)$-th timestamp, the cycle begins anew with this data. The recorded data is first distilled using the data distillation method described in earlier sections, creating a new distilled dataset, $\mathcal{X}'_{t+1}$.

Following this, the distilled data is used to train the current neural network model. The model accuracy is then evaluated, and the model enlargement step is initiated if necessary. This cyclical process ensures consistent improvement and adaptation of the model to new data and conditions over time.

4.2.9 Summarized Process and Advantages

The incremental learning process can be summarized as follows:

1. **Data Distillation:**
$$\mathcal{X}'_{t+1} = \text{Distill}(\mathcal{X}_{t+1}) \tag{4.8}$$
Distill the current dataset $\mathcal{X}_{t+1}$ into a compact representation $\mathcal{X}'_{t+1}$ that captures the essential information.

2. **Neural Network Training:**
$$\mathcal{M}_{t+1} = \text{Train}(\mathcal{M}_t, \mathcal{X}'_{t+1}) \tag{4.9}$$
Train the neural network $\mathcal{M}_t$ on the distilled dataset $\mathcal{X}'_{t+1}$ to produce the updated model $\mathcal{M}_{t+1}$.

3. **Model Accuracy Validation:**
$$A_{t+1} = \text{Validate}(\mathcal{M}_{t+1}, \{\mathcal{T}_{t+1}\}_{t=1}^{n}) \tag{4.10}$$

Validate the updated model $\mathcal{M}_{t+1}$ on all test sets $\{\mathcal{T}_{t+1}\}_{t=1}^{n}$ to ensure it maintains accuracy across previously seen and new data.

4. **Model Enlargement (if necessary):**

$$\mathcal{M}'_{t+1} = \begin{cases} \mathcal{M}_{t+1} + \Delta\mathcal{M}_{t+1} & \text{if } A_{t+1} < A_{\text{standard}} \\ \mathcal{M}_{t+1} & \text{otherwise} \end{cases} \tag{4.11}$$

If the validation accuracy A_{t+1} falls below the predefined threshold A_{standard}, enlarge the model $\mathcal{M}_{t+1}$ to $\mathcal{M}'_{t+1}$ and retrain.

5. **Data Recording and Iteration:** Record the next dataset $\mathcal{X}_{t+2}$, and repeat the process.

Algorithm 2 Incremental Learning with Data Distillation and Model Enlargement

Input: Initial model $\mathcal{M}_0$, initial data $\mathcal{X}_1$, test sets $\{\mathcal{T}_{t+1}\}_{t=1}^{n}$, accuracy threshold A_{standard}
for each timestamp $t \in \{1, 2, \ldots, N\}$ **do**
    Data Distillation: $\mathcal{X}'_{t+1} = \text{Distill}(\mathcal{X}_{t+1})$
    Train Model: $\mathcal{M}_{t+1} = \text{Train}(\mathcal{M}_t, \mathcal{X}'_{t+1})$
    Validate Model: $A_{t+1} = \text{Validate}(\mathcal{M}_{t+1}, \{\mathcal{T}_{t+1}\}_{t=1}^{n})$
    if $A_{t+1} < A_{\text{standard}}$ **then**
        Enlarge Model: $\mathcal{M}'_{t+1} = \mathcal{M}_{t+1} + \Delta\mathcal{M}_{t+1}$
        Retrain model: $\mathcal{M}_{t+1} = \text{Train}(\mathcal{M}'_{t+1}, \mathcal{X}'_{t+1})$
    end if
    Record Next Dataset: $\mathcal{X}_{t+2}$
end for
Output: Trained model $\mathcal{M}_{N+1}$

By repeating this cyclic process for each timestamp t, this methodology ensures the neural network's consistent growth and adaptability to new data and conditions over time. It maintains a balance between resource utilization and model performance, making it an effective solution for continuous learning scenarios on resource-constrained devices.

As illustrated in Algorithm 2, this method offers significant advantages over classical incremental or continuous learning approaches. It requires significantly fewer computational resources since training is performed on distilled datasets, leading to faster convergence. Moreover, it eliminates the need to store the model throughout the entire training period, as it can be quickly reconstructed using the distilled data. This approach also allows for dynamic adaptation of model complexity to match the increasing complexity of the data. Initially, a simpler model suffices when data complexity is low, and as additional steps are introduced, the model is progressively enlarged, an adaptation that is not feasible with many other methods.

4.3 Case Studies: Incremental Learning in Edge ML

In this section, We delve into the specifics of our experiments, shedding light on the chosen datasets and the experimental setup.

4.3.1 Dataset Descriptions

The datasets utilized in our experiments encompass a variety of data types, including image, audio, and sensor data, to evaluate performance across different levels of complexity. These datasets were selected to ensure a comprehensive evaluation of the proposed methods across diverse real-world applications, including vision-based recognition, speech processing, and time-series analysis. Detailed descriptions of each dataset are provided in Appendix A. Below is an overview of the datasets used:

1. **MNIST:** The MNIST dataset is a standard benchmark for handwritten digit recognition. [106] A complete description of this dataset can be found in Appendix A.0.1.

2. **CORe50:** The CORe50 dataset is tailored for continual object recognition tasks, containing diverse object classes and challenging variations. [109] For more details, refer to Appendix A.0.5.

3. **Speech Commands:** The Speech Commands dataset is a benchmark dataset for keyword spotting systems. [77] Additional information about this dataset is available in Appendix A.0.4.

4. **ESC-50:** The ESC-50 dataset is designed for environmental sound classification, providing a diverse set of audio recordings. [110] See Appendix A.0.6 for a comprehensive description.

5. **HAR (Human Activity Recognition):** The Human Activity Recognition (HAR) dataset captures sensor data from daily activities, offering a challenging task for activity recognition. [111] Details about this dataset can be found in Appendix A.0.7.

4.3.2 Experimental Setup

In all experiments, We use a convolutional neural network which was recently used in different scenarios involving distilled data (see e.g. [108]). Networks with depths of 2,3 and 4 blocks of convolutional layers are considered, each containing 8, 64, and 128 kernels, respectively (called ConvNetD2, ConvNetD3 and ConvNetD4). After each convolutional layer, a group normalization layer, ReLU activation function and a 2×2 average pooling with stride 2 follow. In order to model different complexities, We choose suitable value $D \in \{2, 3, 4\}$. This architecture is commonly used for data distillation benchmarks. Even though the data distillation process depends on the model architecture, We find little differences in performance w.r.t. the used model in the distillation, i.e. a ConvNetD2 performs similarly on distilled data generated on ConvNetD2 and on

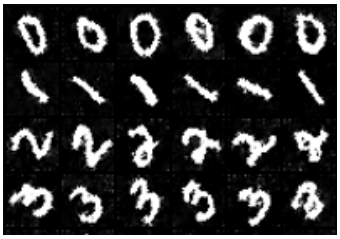

Figure 4.2: Illustrative Examples of Distilled Images in Step 3 of the Continuous Learning Process (MNIST Dataset)

distilled data generated on ConvNetD4. Therefore, We always distill the data w.r.t. the biggest model ConvNetD4. This also allows the transition from a smaller to a larger model during incremental learning. In training, We apply DSA in the distillation process as well as a learning rate scheduler and stochastic gradient descent equipped with a momentum of 0.9 and a weight decay of $5e-4$. Another reason to choose the relatively small networks ConvNetD2, ConvNetD3 and ConvNetD4 to simulate realistic Edge ML scenarios at the expense of absolute accuracy for the larger datasets. In our standard setup, We use a learning rate of 0.01, the Adam Optimizer and a batch size of 256. Each experiment consists of several steps in which the available dataset changes. We compare four scenarios. For the proposed method, We assume access to the currently available subsets at each step, as well as to the distilled images of all previously seen subsets. For the training with distilled data, two different cases are of interest. One consists of training only on the biggest ConvNetD4 model, which is also the one used for distillation. This is compared to the proposed adaptive model size scenario in which the network size increases with the number of steps, starting with the smallest ConvNetD2 network and, if necessary, upgrading to bigger networks as the training progresses. The needed flops for one epoch are compared for each training. For the baseline, only the data subset for the current step is available. This is the *catastrophic forgetting scenario*. We also compare our method to the fully trained (cumulative) scenario, where all current and past subsets are completely available for training, which is of course the ideal case. After each step, We test the model on the entire test set, if nothing else is mentioned. The exact results can be seen in Table I-V.

MNIST

We use the MNIST dataset for handwritten digits to simulate a continuous learning scenario for changing data distribution in 10 steps. The first step is standard training. In each other steps We modify the classes by rotating the images clockwise by a random angle: in step $i \in \{2, \ldots, 10\}$ each training and test image gets rotated by an angle (in degree)

$$\alpha_{train} \sim \mathrm{U}([20(i-2), 20(i-1)]), \quad \alpha_{test} \sim \mathrm{U}([0, 20(i-1)])$$

respectively, where U is the uniform distribution, see Figure 4.2. This method causes some difficulties. For instance, the numbers 6 and 9 can not be distinguished for angles close to 180°. We distill 10 images per class from these randomly rotated training images. We omit DSA in this experiment, since it also contains rotation as an augmentation technique. In total, this is about 1.7% of the data compared to the entire training data set.

CORe50

We preform an incremental learning scenario for the CORe50 dataset on object level classification (50 classes) and the NIC-scenario (new instances and new classes in every step). This is a rather difficult but highly realistic incremental learning setup, since new objects as well as new backgrounds are added each step. The first step contains of 10 classes in one session. Each other step provides 5 classes of one session, resulting in 79 overall learning steps (3 sessions are used for testing). All images are rescaled to a size of 64x64. We generate one distilled image for one class in one session leading to 400 distilled images in total which is about 0.3% is the entire dataset. Our method preserves over 78% test accuracy of the fully (cumulative) trained network and prevents catastrophic forgetting. This also outperforms other popular incremental learning methods for CORe50 like CWR which maintains about 50% and does not allow adaptive model size (see [109]) Best results are archived by training a new network in every step only on the distilled images. There are at most 400 distilled images in each step, hence the training is highly efficient. The mixed training with current real and previous distilled images show more fluctuations and slightly worse results.

Speech Commands

In this experiment, We performed standard class incremental learning for the audio dataset Speech Commands with 35 steps. After converting the audio data into mel frequency and a spectrogram, We passed it as image data with size $51x81$ into the same convolutional network as in the prior experiments. We distilled 20 images per class, which results in about 0.8% of the size of the entire set.

HAR

The Human Activity Dataset contains sensor data which We again converted into mel spectrograms and passed it as images of size $26x31$ pixels into the convolutional network. The class incremental learning consists of 8 steps. In this experiment, 10 IPC are used which is about 4.68% of the data.

CIFAR10

Finally, We consider the classical CIFAR10 dataset in the class incremental learning scenario with 10 steps. We distill 50 IPC, leading to 1% of the size of the original dataset.

4.4 Performance Metrics and Discussion

Within the results section, We provide a comprehensive visual analysis through Figures 4.3-4.7, which illustrates the performance metrics of various models in relation to their computational complexity, measured in FLOPs.

4.4.1 Explanation of Terms

- **Baseline**: Represents the model's performance when trained on the entire dataset without incremental learning. This serves as a reference point for comparing the efficiency and effectiveness of our incremental learning approach.
- **Largest Model**: This refers to a static, large model trained using the incremental learning approach but without adapting the model size.
- **Adaptive Model**: A dynamic model that adjusts its size during training, in line with the complexity of the task at each incremental learning step.
- **Catastrophic Forgetting**: A scenario where the model is trained only on the current data subset, without access to previous data, leading to a rapid decline in performance on previously learned tasks.

The results of our experiments conducted on various datasets are detailed in this section. They are summarized in Tables 4.1 - 4.7, covering CIFAR10, CORe50, Speech Commands, MNIST, and HAR respectively. The outcomes are measured in terms of the end accuracy (accuracy achieved at the final stage), the average accuracy over time (across all steps), and the computation resources required (expressed as a percentage of floating point operations, or FLOPs).

4.4.2 Results

In this section, results from various models across CIFAR10, CORe50, Speech Commands, MNIST, and HAR are summarized. Performance metrics, including end accuracy, average accuracy, and computational resource usage (FLOPs), are compared, with a focus on the impact of catastrophic forgetting.

CIFAR10

As depicted in Table 4.1, our baseline model achieved an end accuracy of 85.4%, averaging 47.8% accuracy over all steps and requiring 100% of the FLOPs. The largest model showed a decrease in performance with an end accuracy of 57.7% and an average accuracy of 35.6%, while still requiring the same amount of computational resources. Notably, the adaptive model demonstrated similar performance to the largest model but at a significantly lower computational cost, achieving an end accuracy of 58.2% and an average accuracy of 35.9% using only 43% of the FLOPs. The catastrophic forgetting scenario, where only the data subset for the current step is available, resulted in an end and average accuracy of 10%, indicating a considerable decline in model performance.

	End Acc.	**Average Acc.**	**Needed FLOPs**
Baseline	85.4 %	47.8 %	100 %
Largest model	57.7 %	35.6 %	100 %
Adaptive Model	58.2 %	35.9 %	43%
Catastrophic Forgetting	10 %	10 %	100 %

Table 4.1: Results for CIFAR10 Dataset: Comparing end accuracy, average accuracy over incremental learning steps, and computational resource requirements (FLOPs) for different models.

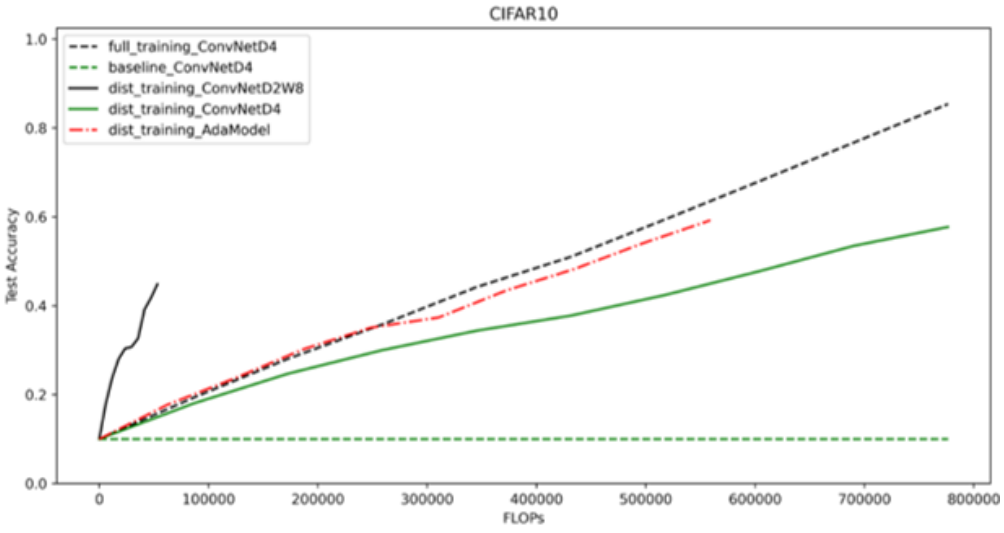

Figure 4.3: Comparison of models based on their performance plotted against the number of FLOPs for the CIFAR10 Dataset.

CORe50

As can be seen in Table 4.2, the baseline model had an end accuracy of 55.3% and an average accuracy of 36.5% over time. The largest and adaptive models exhibited slightly lower performances, but the adaptive model required only half the computational resources (49% of FLOPs). The catastrophic forgetting model fared the worst, with an end accuracy of 3.7% and an average accuracy of 3.6%.

	End Acc.	**Average Acc.**	**Needed FLOPs**
Baseline	55.3 %	36.5 %	100 %
Largest model	44.2 %	28.9 %	100 %
Adaptive Model	43.2 %	26.1 %	49%
Catastrophic Forgetting	3.7 %	3.6 %	100 %

Table 4.2: Results for CORe Dataset: Comparing end accuracy, average accuracy over incremental learning steps, and computational resource requirements (FLOPs) for different models.

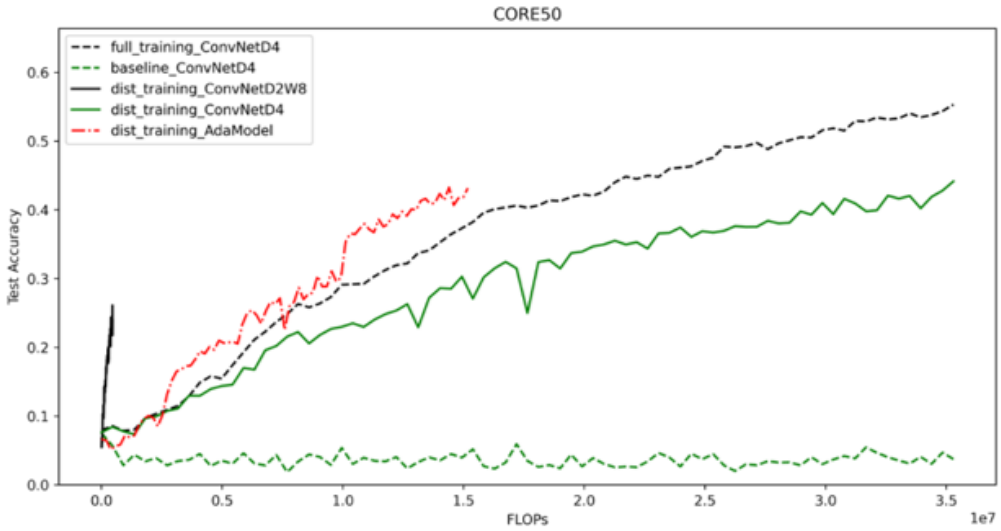

Figure 4.4: Comparison of models based on their performance plotted against the number of FLOPs for the CORe Dataset.

Speech Commands

In the Speech Commands experiment (Table 4.3), the baseline model attained a high end accuracy of 95% and averaged 45.6% over time. The largest and adaptive models reached around 70% in terms of end accuracy, with the adaptive model achieving this using only 36% of the FLOPs. Once again, the catastrophic forgetting model performed poorly, with an end accuracy of 3.8% and an average accuracy of 2.9%.

	End Acc.	**Average Acc.**	**Needed FLOPs**
Baseline	95 %	45.6 %	100 %
Largest model	70.8 %	35 %	100 %
Adaptive Model	70.4 %	33.7 %	36 %
Catastrophic Forgetting	3.8 %	2.9 %	100 %

Table 4.3: Results for Speechcommands Dataset: Comparing end accuracy, average accuracy over incremental learning steps, and computational resource requirements (FLOPs) for different models.

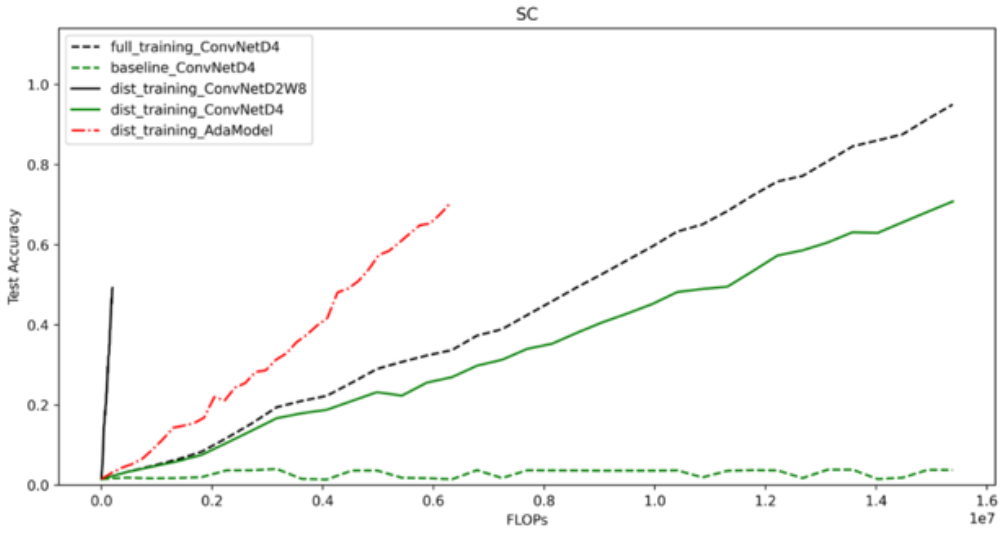

Figure 4.5: Comparison of models based on their performance plotted against the number of FLOPs for the SC Dataset.

MNIST

Table 4.4 illustrates the MNIST results. The baseline model achieved an impressive end accuracy of 98.8% and an average accuracy of 78.8%. The largest and adaptive models demonstrated comparable performances in the 92% range for end accuracy, with the adaptive model achieving this using only 36% of the FLOPs. Interestingly, the catastrophic forgetting model in this case was not as disastrous as in the other scenarios, reaching an end accuracy of 60.1% and averaging 66.2% over time.

	End Acc.	**Average Acc.**	**Needed FLOPs**
Baseline	98.8 %	78.8 %	100 %
Largest model	92.7 %	76.9 %	100 %
Adaptive Model	92.8 %	76.1 %	36%
Catastrophic Forgetting	60.1 %	66.2 %	100 %

Table 4.4: Results for MNIST Dataset: Comparing end accuracy, average accuracy over incremental learning steps, and computational resource requirements (FLOPs) for different models.

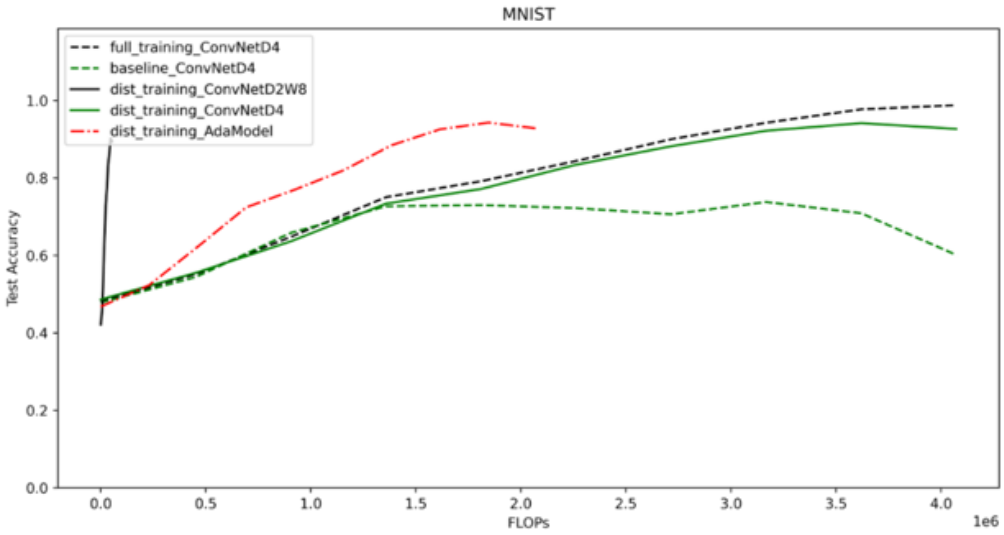

Figure 4.6: Comparison of models based on their performance plotted against the number of FLOPs for the MNIST Dataset.

HAR

As shown in Table 4.5, for the HAR dataset, the baseline model achieved an end accuracy of 83.1% and averaged 43.1% over time. The largest and adaptive models exhibited end accuracies in the mid-60% range, with the adaptive model again reducing computational resources by nearly two-thirds. In this scenario, the catastrophic forgetting model resulted in a significant drop in performance, with an end accuracy of just 15% and an average accuracy of 12.5%.

	End Acc.	**Average Acc.**	**Needed FLOPs**
Baseline	83.1 %	43.1 %	100 %
Largest model	65.1 %	37.3 %	100 %
Adaptive Model	64.4 %	36.9 %	36%
Catastrophic Forgetting	15 %	12.5 %	100 %

Table 4.5: Results for HAR Dataset: Comparing end accuracy, average accuracy over incremental learning steps, and computational resource requirements (FLOPs) for different models.

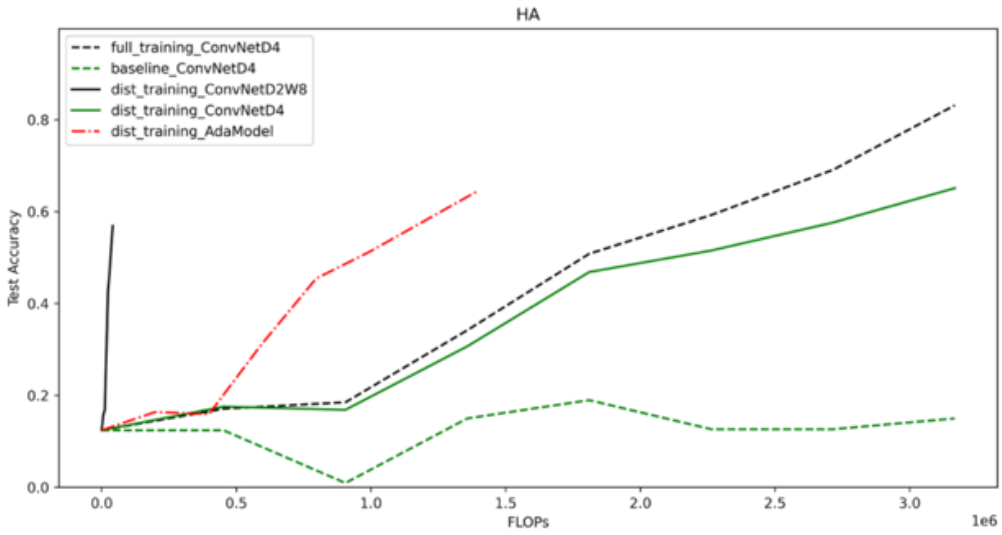

Figure 4.7: Comparison of models based on their performance plotted against the number of FLOPs for the HAR Dataset.

4.4.3 Discussion

In this section, We delve into the findings obtained from our experimentation on five diverse datasets: CIFAR10, CORe50, Speech Commands, MNIST, and HAR. These results elucidate the successful implementation and benefits of the incremental learning approach proposed in this study, mainly focusing on memory efficiency and computational savings.

Incremental Learning and Memory Efficiency

Our experiments demonstrated the successful application of the incremental learning technique, especially in the context of on-device training where memory limitations are often a hindrance. The use of distilled datasets proved beneficial in combating the challenge of catastrophic forgetting typically associated with continuous learning. Remarkably, the memory overhead required for this technique was a mere 1% of the total size of the original dataset. This reveals the potential of our proposed method in situations where device memory is a significant concern.

Efficiencies of the Adaptive Model Size

A unique aspect of our method is the adaptive expansion of our models, which helps to alleviate the demands on memory and computational resources. By incrementally increasing the model size as per the task requirements, We could avoid deploying an overly large model from the outset.

When examining the FLOPs required across the training process, our method demonstrated considerable savings. In the Speech Commands and HAR datasets, We only utilized 36% of the FLOPs required by a fixed model. For the CIFAR10 dataset, this requirement dropped to 43%, and even for the computationally heavy CORe50 dataset, our method needed only 49% of the FLOPs. These reductions directly correlate with energy and time savings, making our technique particularly valuable for on-device training where these resources are scarce.

Further, our results showed minimal, if any, sacrifice in accuracy despite these significant savings. On average, the CIFAR10 dataset experienced a slight 0.3% improvement in accuracy over the entire training process, while the CORe50 dataset recorded a marginal average loss of 2.8% in accuracy. These outcomes suggest that the proposed method offers an effective balance between resource utilization and model performance.

In summary, our proposed method not only handles the inherent constraints of on-device training but also showcases efficient resource utilization while preserving model accuracy.

Chapter 5

Computational Efficiency in Edge ML: Advancing Sparse Backpropagation Methods

In this chapter, methods to advance computational efficiency in Edge ML through sparse backpropagation techniques are investigated. The principles of sparse backpropagation, along with the TinyPropv2 algorithms, are discussed to illustrate their benefits in reducing computational load during on-device training.

5.1 Fundamentals of Sparse Backpropagation

Sparse backpropagation optimizes the training of deep neural networks (DNNs) by selectively updating model parameters, thus reducing computational requirements. This technique is especially beneficial for Edge ML applications, where device constraints limit traditional training methods. 2.4

5.1.1 Principles of Sparse Backpropagation

Sparse backpropagation is a technique designed to optimize the training of deep neural networks (DNNs) by selectively updating only a subset of model parameters during the backpropagation process [112]. This selective updating is particularly advantageous for Edge Machine Learning (Edge ML) applications, where computational resources such as processing power, memory, and energy are limited. Unlike traditional backpropagation, which updates all the parameters in the model, sparse backpropagation focuses on the most significant weights, effectively reducing the overall computational load while maintaining model accuracy.

The underlying principle of sparse backpropagation is based on the observation that not all parameters contribute equally to the learning process. In most neural networks, a small number of weights have much larger gradients than the others during the training phase. These gradients signify the direction and magnitude of the necessary changes in the weights to minimize the model's loss. Sparse backpropagation leverages this by identifying and updating only the parameters with the largest gradients, thus speeding up the training process and reducing resource consumption [113].

The selective updating of weights is usually controlled through a thresholding mechanism, where only the top-k gradients (those with the highest magnitudes) are retained

for weight updates [95]. The remaining gradients are set to zero, effectively reducing the number of weight adjustments and the computational overhead associated with backpropagation. This method is particularly well-suited for deep networks deployed on edge devices, where conserving computational resources is critical.

In practice, sparse backpropagation can be implemented in several ways:

- **Top-k sparsification:** In this approach, only the k largest gradients are selected for updating during each backpropagation step. This method ensures that only the most significant weights are adjusted, reducing the overall number of computations.
- **Gradient pruning:** Gradients below a certain magnitude threshold are pruned (set to zero) during the training process. This not only reduces the computational load but also promotes model sparsity, which can enhance the efficiency of model inference on edge devices.
- **Adaptive sparsification:** The sparsity level (the fraction of gradients to be updated) is dynamically adjusted based on the model's training progress or resource constraints. This allows the algorithm to adapt to varying conditions, such as changes in the available computational resources or the complexity of the current training task.

5.1.2 Relevance in Edge ML

Sparse backpropagation represents a pivotal advancement in the realm of Edge Machine Learning (Edge ML), addressing several critical challenges inherent to deploying deep learning models on resource-constrained devices. This section elucidates the significance of sparse backpropagation techniques, particularly their adaptability and efficiency, in the context of Edge ML.

- **Addressing Resource Limitations:** Edge ML devices, such as Micro-Controller Units (MCUs), are often constrained by limited memory, energy resources, and computing power. Sparse backpropagation techniques optimize the training process by reducing the computational load, thus making the most out of the available resources.
- **Enhancing On-device Learning:** By enabling more efficient model training directly on the device, sparse backpropagation methods facilitate on-device learning. This approach not only improves data security by minimizing the need to share information with external entities but also reduces latency and energy costs associated with data transmission and cloud-based computations.
- **Dynamic Adaptability:** The adaptive nature of the proposed sparse backpropagation algorithms, such as TinyProp, allows for dynamic adjustment of the backpropagation ratio. This adaptability ensures that the training process is optimized in real-time based on the model's current state and the available resources, enhancing the efficiency and effectiveness of on-device training.

- **Mitigating Overfitting and Ensuring Model Robustness:** Sparse backpropagation methods contribute to the model's robustness by preventing overfitting. By selectively updating weights and incorporating mechanisms to skip redundant training steps, these techniques help maintain the model's generalizability to new data.
- **Contributions to the Edge ML Ecosystem:** The introduction of sparse backpropagation techniques, including the innovative use of Grad-CAM visualizations for data retention decisions, marks a significant contribution to the Edge ML ecosystem. These methods not only address the computational and storage challenges but also open up new possibilities for real-time, efficient learning on embedded systems.

In summary, sparse backpropagation techniques are instrumental in advancing the capabilities of Edge ML by providing solutions that are both resource-efficient and adaptable to the dynamic requirements of embedded systems. Their development and implementation represent a step forward in realizing the full potential of on-device machine learning.

5.2 Efficient Backpropagation through TinyProp

Deep learning, especially in the context of deep neural networks (DNNs), has shown remarkable success in various applications. However, the training process, which involves forward and backward propagation, can be computationally intensive. This section introduces the standard forward and backward propagation methods and explains the concept of sparse backpropagation, the basis for the original TinyProp algorithm. We then detail the advancements made in our enhanced version, TinyPropv2.

5.2.1 From Fixed Sparsity to Adaptive Sparsity: Evolution and Advantages

The shift from fixed to adaptive sparsity marks a critical step in improving the efficiency and effectiveness of on-device training. Before detailing the underlying motivations and design choices, we first examine the practical limitations of fixed sparsity approaches.

Motivation for Adaptive Sparsity

Fixed sparsity involves selecting a predetermined percentage of gradients to update across all layers and training iterations, irrespective of the data characteristics or training progress. While this approach simplifies implementation, it suffers from several limitations:

- **Underutilization of Resources:** During early training stages, more gradients typically contribute to meaningful updates. Fixed sparsity may fail to leverage this, slowing convergence.

- **Wasted Resources:** In later stages, when many gradients have negligible contributions, fixed sparsity may allocate updates unnecessarily, leading to inefficiency.

In contrast, adaptive sparsity dynamically adjusts the proportion of gradients updated based on error signals or other layer-specific metrics. This adaptivity focuses computational resources where they are most needed, leading to better resource utilization and faster convergence.

From Fixed to Adaptive Sparsity: The Evolution of the Idea

The transition from fixed to adaptive sparsity was motivated by the need to address inefficiencies in the fixed approach. Key observations and insights include:

Observations in Fixed Sparsity

- **Uniformity Problem:** Fixed sparsity applies the same update ratio across all layers and epochs, ignoring the unique characteristics of layers and the dynamic nature of training. This often results in poor resource utilization.
- **Lack of Flexibility:** Fixed sparsity cannot adapt to changing training dynamics, such as diminishing gradient magnitudes over time or varying convergence rates across layers.

Insights Driving Adaptivity

- **Error-Driven Learning:** Studies revealed that only a small subset of weights significantly influences learning during each training step.
- **Dynamic Gradient Behavior:** Research on gradient distributions highlighted that important gradients shift over time and differ across layers, making static sparsity levels ineffective.

Advantages of Adaptive Sparsity

Adaptive sparsity offers several benefits over fixed sparsity:

1. **Layer-Specific Adaptivity:**
   - Different layers contribute unequally to learning at various training phases. For example, earlier layers may converge faster, requiring fewer updates, while deeper layers may need more updates to refine complex features.
   - Adaptive sparsity dynamically adjusts gradient updates in each layer based on its importance and contribution to overall loss reduction.
2. **Training Phase Adaptivity:**
   - In early training stages, gradients are larger and more distributed, necessitating higher update thresholds. As training progresses, gradients diminish in magnitude, and adaptive sparsity focuses on fewer but more critical updates.

- Fixed sparsity fails to account for this dynamic change, leading to inefficient resource allocation.

3. **Error-Driven Optimization:**
   - Adaptive sparsity leverages error signals or gradient magnitudes to identify the most important updates.
   - This ensures efficient learning by dynamically adjusting sparsity levels based on training errors or layer-specific characteristics.
4. **Energy and Memory Efficiency:**
   - Fixed sparsity may over-allocate resources to layers or epochs that do not need significant updates, increasing energy consumption.
   - Adaptive sparsity reduces updates where they contribute less, making it ideal for resource-constrained environments like MCUs.
5. **Improved Convergence Properties:**
   - Adaptive sparsity ensures responsiveness to changes in data distribution or error signals, enhancing convergence speed and accuracy.
   - Fixed sparsity may result in slower convergence or even stagnation if the chosen sparsity level is inappropriate for a particular training phase.

The evolution from fixed to adaptive sparsity represents a significant advancement in neural network training optimization. By leveraging dynamic metrics to allocate resources efficiently, adaptive sparsity ensures gradient updates are focused where they are most impactful. This adaptivity leads to faster convergence, improved generalization, and greater suitability for resource-constrained environments and real-time applications.

5.2.2 Forward and Backward Propagation

The processes of forward and backward propagation form the backbone of DNN training. Forward propagation computes the output of the network given an input, while backward propagation updates the network's weights based on the error of its predictions. A comprehensive explanation of these methods, including the mathematical formulations, is provided in Appendix B.1.

5.2.3 Sparse Backpropagation

Sparse backpropagation is an optimization technique that aims to reduce the computational complexity of the standard backpropagation algorithm. Instead of updating all the weights in the network, sparse backpropagation updates only a subset of them, specifically those with the largest gradients. This approach is based on the observation that only a few weights, which have the most significant gradients, contribute the most to the learning process. [112]

Gradient Sparsity

Given a gradient vector δ_a^l for layer l, the top-k elements based on their magnitude can be represented as:

$$\text{top}(\delta_a^l, k) \tag{5.1}$$

For instance, for a gradient vector $v = [1, 2, 3, -4]^T$, the top-2 elements would be represented as $\text{top}(v, 2) = [0, 0, 3, -4]^T$.

Approximate Gradient

The approximate gradient is then computed by retaining only the top-k elements and setting the rest to zero:

$$\hat{\delta}_a^l = \begin{cases} \delta_a^l & \text{if } a \in \{t_1, t_2, \ldots, t_k\} \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

This approximation ensures that only the most significant gradients contribute to the weight updates, leading to a reduction in computational effort.

Sparse Gradient Propagation

Using the approximate gradient, the backpropagation is modified as:

$$\hat{\delta}_a^l = \text{top}(\delta_a^l, k) \tag{5.3}$$

$$\hat{\delta}_z^l = \hat{\delta}_a^l \odot f'\left(z^l\right) \tag{5.4}$$

$$\hat{\delta_a^{l-1}} = W^{l-1}\hat{\delta}_z^l \tag{5.5}$$

Where f' is the derivative of the activation function.

By employing this sparse approach, the backpropagation algorithm becomes more efficient, especially for deep networks with a large number of parameters.

5.2.4 The TinyProp Algorithm

The TinyProp algorithm enhances the efficiency of training deep neural networks (DNNs) by implementing a dynamic sparse backpropagation approach. This method is particularly effective for on-device learning on tiny, embedded devices such as low-power microcontroller units (MCUs). [40]

TinyProp Adaptivity Approach

The core innovation of TinyProp lies in its adaptivity, where the algorithm dynamically calculates the proportion of gradients to be updated for each layer during training. This adaptivity is based on the local error in each layer, enabling the algorithm to focus computational resources more effectively.

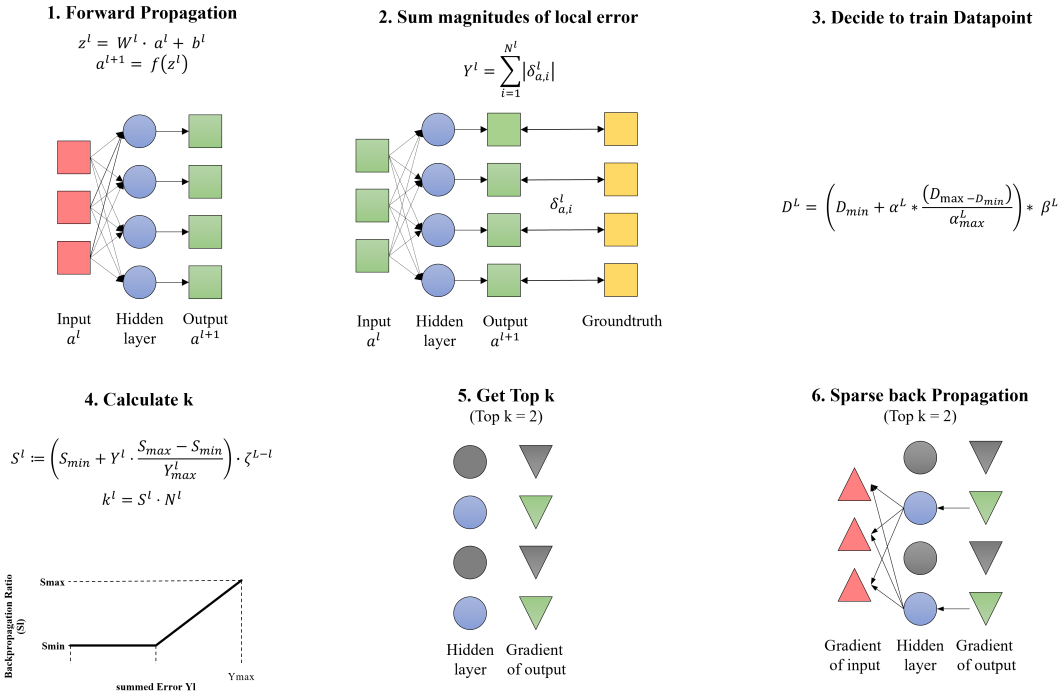

Figure 5.1: Operational Workflow of TinyPropv2: The process begins with (1) performing the forward pass to compute the output. This is followed by (2) calculating the loss function and accumulating the local errors. Subsequently, (3) a decision is made on whether to train the datapoint based on the computed error. Next, (4) the optimal number of gradients to update, denoted as 'local k,' is determined from the aggregated error. (5) The algorithm then identifies the top 'k' gradients that will be updated. Finally, (6) these selected gradients undergo the sparse backpropagation process, completing the training step.

Local Error Vector and Total Error: The adaptivity mechanism in TinyProp uses the local error vector $\delta_{a,i}^l$ of each layer to gauge the layer's contribution to the overall error. The total error characteristic of the layer Y^l is computed as the sum of the absolute values of these error components:

$$Y^l = \sum_{i=1}^{N^l} \left|\delta_{a,i}^l\right| \tag{5.6}$$

Adaptive Error Propagation Rate: TinyProp introduces an adaptive error propagation rate S^l, which reflects the training progress and is calculated as a function of the total error:

$$S^l = S_{\min} + Y^l \frac{S_{\max} - S_{\min}}{Y^l_{\max}} \tag{5.7}$$

where $S_{\max}$ and $S_{\min}$ are user-defined bounds for the error propagation rate.

TinyProp Damping Factor: To address the computational intensity in larger DNN layers, TinyProp incorporates a damping factor $\zeta(l)$ that reduces the error propagation rate in a layer-dependent manner:

$$S^l = \left(S_{\min} + Y^l \frac{S_{\max} - S_{\min}}{Y^l_{\max}} \right) \zeta^{-l+L} \tag{5.8}$$

This factor allows for reduced computational effort in layers that typically require less training, such as the initial layers of the network.

Computing the Adaptive Top-k: With the calculated error propagation rate S^l, TinyProp determines the number of gradients to update in each layer adaptively:

$$k^l = S^l \cdot N^l \tag{5.9}$$

TinyProp Backpropagation Algorithm: The backpropagation step in TinyProp is then conducted using the adaptive top-k approach:

$$\begin{aligned} \hat{\delta}^l_a &= \text{top}(\delta^l_a, k^l) \\ \hat{\delta}^l_z &= \hat{\delta}^l_a \odot f'\left(z^l\right) \\ \hat{\delta^{l-1}_a} &= \left(W^{l-1}\right)^T \cdot \hat{\delta}^l_z \end{aligned} \tag{5.10}$$

Through this methodology, TinyProp efficiently manages computational resources while maintaining the efficacy of the learning process, making it highly suitable for embedded applications.

5.2.5 The Enhanced TinyProp Algorithm

Building upon the original TinyProp algorithm, TinyPropv2 introduces an additional layer of decision-making to potentially skip entire training steps when beneficial 5.1. The primary motivation for incorporating this additional decision layer stems from the need to enhance computational efficiency without sacrificing the model's accuracy. Traditional training methods often expend significant computational resources on processing every data point, regardless of its actual impact on the model's learning. TinyProp, by contrast, intelligently identifies and focuses on data points that substantially contribute to the learning process. This targeted approach ensures that computational efforts are allocated more judiciously, leading to a more efficient training cycle. This approach further reduces computational effort while maintaining the balance between efficiency and accuracy.

Decision Mechanism for Training Data Points

The decision to train a specific data point in TinyProp is based on a novel decision metric, D, which assesses the necessity of performing backpropagation for that data point:

$$D^L = \left(D_{\min} + \alpha^L \frac{D_{\max} - D_{\min}}{\alpha^L_{\max}} \right) \beta^L \tag{5.11}$$

Where:

- D^L is the decision metric for the last layer L.
- $D_{\min}$ and $D_{\max}$ are the minimum and maximum thresholds for the decision metric.
- α^L is a factor based on the current state of training at layer L.
- β^L is a scaling factor to adjust the sensitivity of the decision metric.

If D^L exceeds a certain threshold, such as 0.5, backpropagation is performed; otherwise, it is skipped. This selective approach prioritizes data points that are more impactful for learning, enhancing efficiency.

Threshold Determination: The threshold against which D^L is compared is not arbitrarily set but is carefully chosen based on empirical observations and domain-specific requirements. For instance, a threshold value of 0.5 is commonly used as a starting point. However, this threshold can be adjusted according to the characteristics of the dataset and the specific learning goals.

- **Empirical Tuning:** The threshold is often empirically tuned during the preliminary stages of model training. This involves experimenting with different threshold values and observing their impact on the model's performance and training efficiency.
- **Dataset Sensitivity:** The optimal threshold may vary depending on the dataset's complexity and the nature of the data points. For example, datasets with more noise or variability might require a different threshold approach compared to more uniform datasets.
- **Performance Metrics:** The decision on the threshold also considers the balance between training efficiency and accuracy. A higher threshold might speed up training but at the cost of accuracy, and vice versa.

Benefits of TinyPropv2 Over TinyProp

TinyPropv2 extends the capabilities of the original TinyProp algorithm by:

- **Reducing Computational Load:** By intelligently deciding whether to perform backpropagation for each data point.

- **Enhanced Adaptability:** Offers more refined control over the training process, suitable for various types of datasets and learning scenarios.
- **Resource Optimization:** Particularly beneficial for MCUs and embedded devices where computational resources are limited.

5.2.6 Pseudocode for the TinyPropv2 Algorithm

The following pseudocode outlines the steps involved in the TinyProp and TinyPropv2 algorithms, highlighting the dynamic sparse backpropagation approach and the decision mechanism unique to TinyPropv2.

Algorithm 3 TinyPropv2 Algorithm

Require: Training data, Network architecture
Ensure: Trained network weights
for each training epoch **do**
  for each data point **do**
    /* TinyProp Decision Mechanism */
    Compute decision metric D^L (TinyProp)
    if D^L exceeds threshold (TinyProp) **then**
      /* End of TinyProp-specific step */
      for each layer l in the network **do**
        Compute forward propagation
        Calculate local error vector $\delta^l_{a,i}$
        Compute total layer error Y^l
        Calculate adaptive error propagation rate S^l
        Compute damping factor adjustment $\zeta(l)$
        Determine adaptive top-k value k^l
        Compute sparse gradient $\hat{\delta}^l_a$
      end for
      for each layer l in backward order **do**
        Apply sparse backpropagation updates
      end for
    else
      Skip backpropagation for this data point (TinyPropv2)
    end if
  end for
end for

5.3 Case Studies: Sparse Backpropagation in Edge ML

This section presents a comprehensive evaluation of the TinyPropv2 algorithm using diverse datasets and experimental setups to assess its effectiveness in improving backpropagation efficiency on resource-constrained edge devices.

5.3.1 Selected Datasets for Validation

The experimental validation of the TinyPropv2 algorithm was conducted using a diverse set of open-source datasets, chosen for their prominence in benchmarking within the machine learning community and their representation of various application domains. Detailed descriptions of each dataset can be found in Appendix A. Below is an overview of the datasets used:

1. **CIFAR-10 and CIFAR-100:** These datasets are fundamental benchmarks for image classification tasks. CIFAR-10 comprises 10 classes, while CIFAR-100 provides finer granularity with 100 classes. Comprehensive descriptions are available in Appendix A.0.2.

2. **Oxford Flowers 102:** This dataset presents a fine-grained classification challenge, capturing diverse flower species. Further details are provided in Appendix A.0.9.

3. **Food-101:** The Food-101 dataset simulates real-world scenarios with noisy and imbalanced data, making it a robust benchmark for image recognition. See Appendix A.0.10 for more information.

4. **Speech Commands:** The Speech Commands dataset serves as a benchmark for keyword spotting and speech recognition. Additional details can be found in Appendix A.0.4.

5. **MNIST:** The MNIST dataset is a classic benchmark in machine learning for handwritten digit recognition. A detailed description is provided in Appendix A.0.1.

6. **Human Activity Recognition (HAR):** The HAR dataset captures sensor data from daily activities, presenting a challenging task for activity recognition. Refer to Appendix A.0.7 for more details.

7. **DCASE2020 Challenge Task 1:** This dataset from the DCASE2020 Challenge focuses on anomaly detection in machine sounds. Details about this dataset are available in Appendix A.0.8.

Each dataset contributes uniquely to evaluating the algorithm's performance across various domains, encompassing challenges such as image recognition, audio processing, and sensor data analysis. The selection ensures a comprehensive validation of the TinyPropv2 method across heterogeneous real-world scenarios.

5.3.2 Models and Architectures

Our experiments leveraged the MobileNetV2 architecture [114], renowned for its efficiency on mobile devices, and a bespoke 5-layer neural network tailored to the dataset modality—employing either 1D or 2D convolutions for time-series and image data, respectively. These models were initially pretrained on the ImageNet dataset to establish a foundational knowledge before being fine-tuned for our specific datasets.

5.3.3 Computational Environment

A critical aspect of evaluating the efficacy of any machine learning algorithm, particularly those designed for on-device deployment, is the computational environment in which the experiments are performed. For our experiments, we selected a controlled computational setting that would reflect the constraints and capabilities of high-performance embedded systems.

Software Framework: We utilized Python as the primary programming language for its widespread adoption in the scientific computing community and its comprehensive suite of machine learning libraries. The experiments were implemented using the PyTorch 2.0.0 framework, benefiting from its dynamic computation graph and efficient memory management for deep neural network training.

Training Protocol: Consistency in the training protocol was maintained across all experiments to ensure comparability. We adopted Stochastic Gradient Descent (SGD) [115] without momentum or weight decay, coupled with a cosine annealing scheduler to modulate the learning rate across 200 epochs. The learning rate was initialized at 0.125 and annealed to zero, with a warm-up phase of 5 epochs. This training policy was selected to mirror typical fine-tuning practices in resource-limited settings, where complex adaptive optimization algorithms may not be feasible.

Evaluation Metric: Accuracy was determined by evaluating the top-1 performance metric on the respective test sets (D_{test}) of each dataset. This metric was chosen for its straightforward interpretability and its common use as a benchmark in classification tasks.

The computational environment was carefully architected to strike a balance between replicating the limitations of embedded devices and ensuring the reproducibility of results. It provided a rigorous testbed for our algorithms and a reliable indicator of their potential performance in real-world applications.

5.4 Performance Evaluation

The evaluation of the TinyProp method involved a dual-focus analysis: first, We assessed the accuracy of the method across various datasets; second, We examined the computational effort required during training. This two-pronged approach enabled us to investigate not only the effectiveness of the model in terms of learning capabilities but also its efficiency, which is crucial for deployment on resource-constrained devices.

5.4.1 Accuracy

Our accuracy assessment revealed that TinyProp competently navigated the trade-off between model complexity and learning accuracy shown in Tab. 5.1. In datasets with higher-dimensional data and more complex structures, such as CIFAR-100 and DCASE2020, TinyProp demonstrated a remarkable capacity to maintain high accuracy levels, rivaling the full training baseline without necessitating extensive computational resources.

In simpler datasets like MNIST, the accuracy advantage of TinyProp over other methods became less pronounced, suggesting that its benefits are more significant in scenarios where the learning task inherently involves more complexity and where discerning the salient features from the data is more challenging.

5.4.2 Computational Effort

One of the notable features of TinyProp is its ability to skip training on certain datapoints as the model becomes more competent. This approach effectively combats overfitting by reducing the training intensity as the model's accuracy improves. Consequently, as the model training progresses and the algorithm identifies less error-prone data, the computational effort required diminishes significantly.

The computational effort analysis, as depicted in the accompanying bar chart, shows that TinyProp rapidly decreases its computational load relative to other methods. Initially, the effort aligns closely with that of Sparse Update and TinyTrain approaches. However, as training progresses and the algorithm becomes more selective in the datapoints it deems necessary to train on, We observe a steeper decline in computational effort for TinyProp.

This behavior underscores the method's adaptability and responsiveness to the learning progress, offering an efficient training process that dynamically adjusts to the model's evolving state of knowledge. It affirms TinyProp's potential as a scalable solution for on-device learning, where computational resources are often at a premium and must be judiciously allocated.

5.4.3 Comparative Analysis

The comparative analysis of computational effort required for different training methods is illustrated in Figure 5.2. As shown, TinyProp starts on par with other methods but as training continues, the effort required for TinyProp decreases more steeply. This is due

to its unique ability to skip redundant datapoint training, thereby reducing unnecessary computations. As a result, TinyProp not only conserves computational resources but also mitigates the risk of overtraining, striking a desirable balance between learning efficiency and model performance.

5.4.4 Implications for On-Device Learning

The implications of these results are significant for on-device machine learning applications. TinyProp's intelligent computation management makes it particularly suited for environments where power and processing capabilities are limited. By minimizing computational overhead without compromising on learning outcomes, TinyProp ensures that devices such as smartphones, IoT sensors, and other embedded systems can perform complex learning tasks autonomously.

The results from this study provide a compelling case for the adoption of TinyProp in on-device learning scenarios. Its dynamic adjustment to the training process not only optimizes the computational load but also enhances the overall longevity and functionality of devices operating in resource-constrained environments.

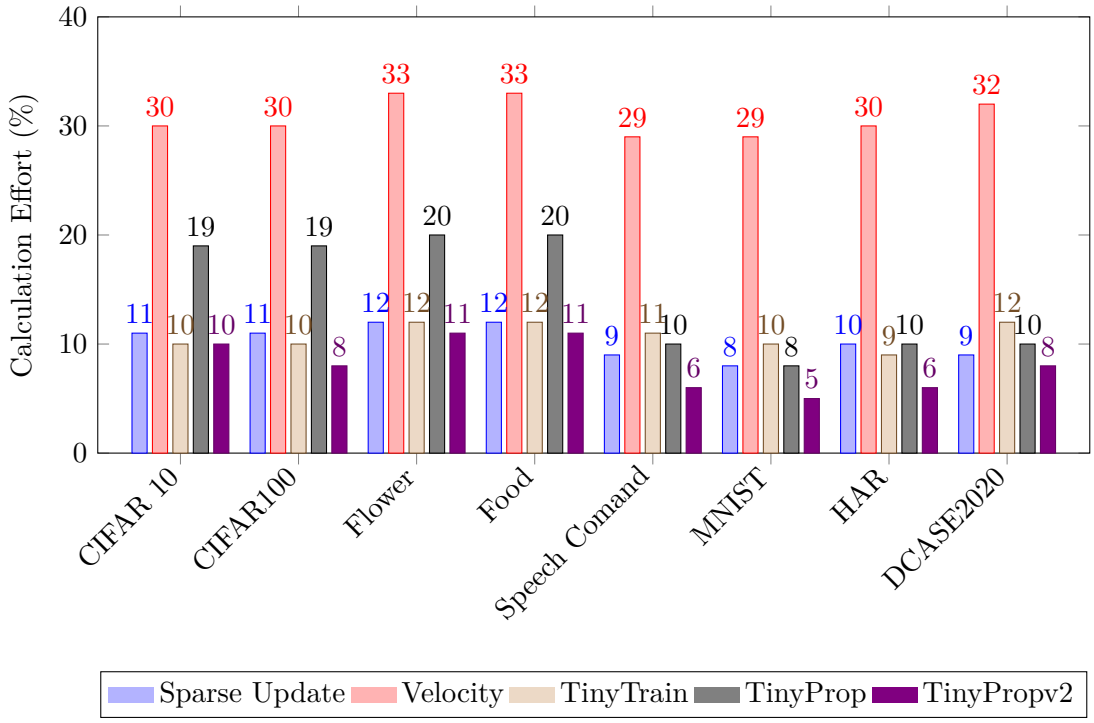

Figure 5.2: Comparative analysis of computational effort required for different training methods across various datasets.

Table 5.1: Accuracy of different methods across various datasets. The Grey highlight result is the best one.

Method	**CIFAR10**	**CIFAR100**	**Flower**	**Food**	**Speech Com.**	**MNIST**	**HAR**	**DCASE**
Model	MNetV2	MNetV2	MNetV2	MNetV2	5 L DNN	5 L DNN	5 L DNN	5 L DNN
Full Training	96.12	80.9	94.01	80.4	96.4	96.6	95.3	98.88
Sparse Update	95.13	78.6	93.77	77.81	93.98	96.41	94.3	97.17
Velocity	95.25	79.46	93.03	79.16	94.51	96.44	94.5	97.96
TinyTrain	94.91	79.51	93.33	79.23	94.6	96.53	95.1	97.97
TinyProp	93.8	78.6	91.6	78.3	93.0	96.32	94.1	97.55
TinyProp	95.3	79.83	93.7	79.1	95.3	96.53	95.2	98.23

Chapter 6

Unified Evaluation and Benchmarking of Proposed Methods

In this chapter, an integrative analysis and comparative study of various techniques for optimizing on ML are conducted. The combined effects of Data Selection, incremental learning, and efficient backpropagation are evaluated. These evaluations provide insights into their collective impact on enhancing the performance and efficiency of Edge ML models in resource-constrained environments.

6.1 Integrating Data Selection, Incremental Learning, and Sparse Backpropagation

This section presents a unified strategy that combines key methods developed in this thesis to address the interrelated challenges of data efficiency, memory constraints, and computational complexity in Edge Machine Learning.

6.1.1 Approach

The integration of Data Selection, incremental learning, and efficient backpropagation techniques provides a comprehensive solution for optimizing Edge Machine Learning (Edge ML) models. Each of these methods addresses specific challenges in Edge ML, and their combination offers synergistic benefits that enhance overall model performance and efficiency.

Data Selection ensures that only the most relevant and informative data points are retained, thereby reducing the computational burden and storage requirements. Incremental learning enables models to adapt continuously to new data without the need for complete retraining, which is crucial for maintaining performance in dynamic environments. Efficient backpropagation, particularly through methods like sparse backpropagation, reduces the computational load during training, making it feasible to train complex models on resource-constrained devices.

The synergistic approach involves leveraging the strengths of each technique to compensate for their individual limitations. For instance, while Data Selection reduces the volume of data, incremental learning ensures that the model can still learn effectively from the selected data points. Efficient backpropagation further optimizes the train-

ing process, making the incremental updates computationally feasible. This integration enhances the adaptability, efficiency, and robustness of Edge ML models.

A significant benefit of this integrated approach is the cascading effect on computational efficiency. By selecting only 50% of the data points through effective Data Selection, the volume of data fed into the training process is halved. Subsequently, employing sparse backpropagation with a sparsity rate of 10% further reduces the computation effort. This chain of benefits results in substantial computational savings, as the reduced dataset requires fewer updates during training, thereby minimizing the overall computational load.

6.1.2 Methodological Integration

The integration of these techniques is methodologically structured to maximize their combined benefits. The process involves several steps:

1. **Data Selection:** Use techniques such as Grad-CAM to assess the importance of data points. Retain only those data points that have a high DRIP score, indicating their relevance for model training.

$$\text{DRIP Score} = \frac{\sum_{i,j} H(I)_{i,j}}{W \times H} \tag{6.1}$$

2. **Incremental Learning:** Implement an incremental learning algorithm that updates the model parameters using the selected data points. This involves computing gradients and updating weights without reprocessing the entire dataset.

$$\theta_{t+1} = \theta_t - \eta \nabla L(\theta_t, D_{\text{new}}) \tag{6.2}$$

3. **Efficient Backpropagation:** Apply sparse backpropagation to reduce the computational load during training. Calculate the top-k gradients and use them to update the model parameters.

$$\hat{\delta}_a^l = \delta_a^l \text{ if } a \in \{t_1, t_2, \ldots, t_k\} \text{ else } 0 \tag{6.3}$$

4. **Adaptive Mechanisms:** Use adaptive techniques to dynamically adjust the proportion of gradients computed during backpropagation based on the local layer error.

$$S^l = \left(S_{min} + Y^l \cdot \frac{S_{max} - S_{min}}{Y^l_{max}} \right) \cdot \zeta^{-l+L} \tag{6.4}$$

$$k^l = S^l \cdot N^l \tag{6.5}$$

This methodological integration ensures that each technique complements the others, resulting in a more efficient and effective learning process. The adaptive nature of the integration allows the model to respond dynamically to changes in data and resource availability, making it suitable for deployment in various Edge ML applications.

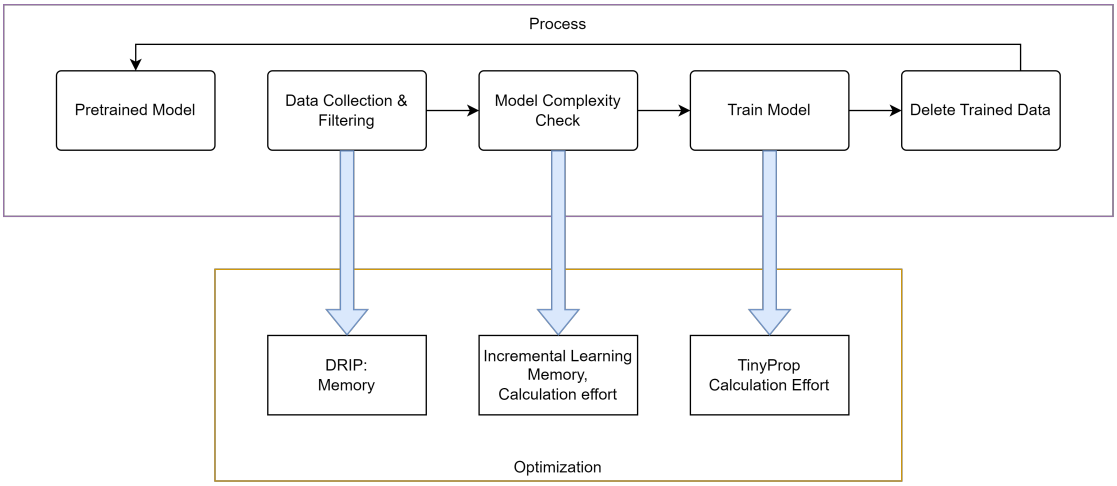

Figure 6.1: Diagram illustrating the integration of Data Selection, incremental learning, and efficient backpropagation in Edge ML.

In summary, the integrated approach leverages the strengths of Data Selection, incremental learning, and efficient backpropagation to optimize Edge ML models for resource-constrained environments. This synergy enhances the overall performance, efficiency, and adaptability of Edge ML applications by creating a chain of benefits that significantly reduce computational and storage requirements.

6.2 Results of the Integration

This section presents the empirical outcomes of the integrated approach combining Data Selection, Incremental Learning, and Sparse Backpropagation. The objective is to evaluate the individual and combined benefits of these techniques on performance metrics such as accuracy, computational cost, and memory usage. The results offer a quantitative assessment of how each method contributes to optimizing Edge ML systems under resource constraints.

6.2.1 Empirical Findings

This subsection provides a detailed overview of the experimental results obtained from benchmarking on diverse datasets. The findings illustrate how the proposed techniques—both individually and collectively—affect system efficiency and learning performance.

Description of Experimental Setup

To ensure consistency and comparability, all experiments were conducted using a standardized computational setup and training protocol. The datasets include MNIST, CIFAR-10, CIFAR-100, Speech Commands, Plant Disease, and HAR, which together span a range of complexities and data modalities. Each method was evaluated under the same conditions to allow direct comparison of results across metrics.

Quantitative Results

The following tables present a comparative analysis of key metrics—accuracy, computational effort, and memory usage—across five configurations: Baseline (no optimization), DRIP (Data Selection), Adaptive Model (Incremental Learning), TinyProp (Efficient Backpropagation), and the full Integrated Approach.

Table 6.1 shows the performance on the MNIST dataset. The integrated approach achieves a significant reduction in computation (1%) and memory usage (22%), with only a minor drop in accuracy.

Metric	**Baseline**	**DRIP**	**Adap. Model**	**TinyProp**	**Integr. Approach**
Accuracy (%)	98.9	98.9	98.7	96.53	96.51
Computation (%)	100	65	36	5	1
Memory Usage (%)	100	61	36	100	22

Table 6.1: Comparison of Performance Metrics Across Different Setups for the Dataset: MNIST

Table 6.2 presents results on the CIFAR-10 dataset, showing that although the integrated method lowers accuracy slightly, it achieves substantial gains in computational and memory efficiency.

Metric	**Baseline**	**DRIP**	**Adap. Model**	**TinyProp**	**Integr. Approach**
Accuracy (%)	96.12	95.1	96.1	95.3	88.8
Computation (%)	100	80	43	10	3
Memory Usage (%)	100	77	43	100	33

Table 6.2: Comparison of Performance Metrics Across Different Setups for the Dataset: CIFAR-10

Table 6.3 continues with CIFAR-100 and confirms consistent performance trends across more complex datasets.

Metric	Baseline	DRIP	Adap. Model	TinyProp	Integr. Approach
Accuracy (%)	93.9	91.5	91.8	91.9	91.0
Computation (%)	100	84	38	8	3
Memory Usage (%)	100	79	38	100	30

Table 6.3: Comparison of Performance Metrics Across Different Setups for the Dataset: CIFAR-100

In Table 6.4, the Speech Commands dataset results demonstrate similar efficiency benefits, with minor accuracy losses compared to the baseline.

Metric	Baseline	DRIP	Adap. Model	TinyProp	Integr. Approach
Accuracy (%)	96.4	95.7	94.4	95.3	93.8
Computation (%)	100	71	36	6	2
Memory Usage (%)	100	67	36	100	24

Table 6.4: Comparison of Performance Metrics Across Different Setups for the Dataset: Speech Commands

Table 6.5 evaluates the Plant Disease dataset, again affirming the effectiveness of the integrated setup in reducing overhead.

Metric	Baseline	DRIP	Adap. Model	TinyProp	Integr. Approach
Accuracy (%)	94.01	94.1	92.9	93.7	92.0
Computation (%)	100	65	40	12	3
Memory Usage (%)	100	60	40	100	24

Table 6.5: Comparison of Performance Metrics Across Different Setups for the Dataset: Plant Disease

Finally, Table 6.6 shows the HAR dataset results, highlighting the consistent efficiency gains delivered by the integrated technique across modalities.

Metric	Baseline	DRIP	Adap. Model	TinyProp	Integr. Approach
Accuracy (%)	95.3	94.5	94.4	95.2	93.7
Computation (%)	100	66	36	6	1
Memory Usage (%)	100	63	36	100	23

Table 6.6: Comparison of Performance Metrics Across Different Setups for the Dataset: HAR

Figure 6.2 visualizes the compounded improvements in resource efficiency achieved through the integrated approach. Each individual optimization contributes to a stepwise reduction in computational effort and memory footprint, underscoring the value of their combination in Edge ML environments.

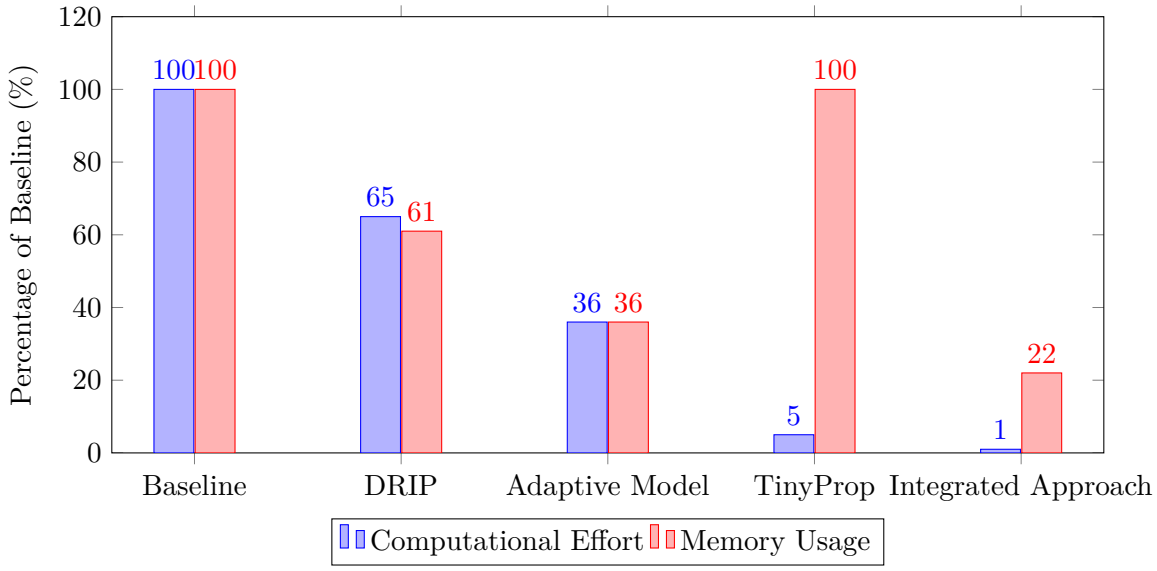

Figure 6.2: Cascading Benefits of Integrated Techniques

This diagram visualizes the cascading benefits of integrating Data Selection, incremental learning, and efficient backpropagation. By reducing the dataset size through Data Selection, fewer data points are processed during training, which, when combined with sparse backpropagation, results in substantial computational savings.

6.2.2 Theoretical Implications

This section explores the broader theoretical insights derived from the experimental findings, focusing on how the combined application of Data Selection, Incremental Learning, and Sparse Backpropagation contributes to improving the efficiency and practicality of Edge ML systems.

Detailed Discussion of Results

The empirical results demonstrate that integrating Data Selection (DRIP), incremental learning (Adaptive Model), and efficient backpropagation (TinyProp) significantly enhances the performance and efficiency of Edge ML models. The cascading benefits of these integrated techniques are evident in the substantial reductions in computational effort and memory usage, as shown in the performance metrics across various datasets.

For instance, in the MNIST dataset, the integrated approach achieved a computation reduction to 1% of the baseline and memory usage down to 22%, while maintaining an

accuracy of 96.51%, only slightly lower than the baseline accuracy of 98.9%. Similar trends were observed in other datasets like CIFAR-10, CIFAR-100, Speech Commands, Plant Disease, and HAR, where the integrated approach consistently reduced computational and memory demands significantly, albeit with minor reductions in accuracy.

These results highlight that:

- **Data Selection (DRIP):** Reduces the dataset size, enabling faster and more efficient training by retaining only the most relevant data points.
- **Incremental Learning (Adaptive Model):** Allows continuous adaptation to new data without the need for complete retraining, ensuring that models stay updated with less computational effort.
- **Efficient Backpropagation (TinyProp):** Minimizes computational load by focusing on the most significant gradients, further enhancing training efficiency.

Analysis of Trade-offs

The integrated approach shows a slight reduction in accuracy compared to using individual techniques. For example, in the CIFAR-10 dataset, the integrated approach achieved an accuracy of 88.8%, compared to 96.12% for the baseline. However, this trade-off is balanced by the significant gains in computational efficiency and memory usage. The integrated approach reduced computation to 3% and memory usage to 33% of the baseline. This balance between performance and resource savings is crucial for Edge ML applications, where computational and memory constraints are critical considerations.

The trade-offs indicate that while some accuracy is sacrificed, the benefits in terms of reduced computational time and memory usage make the integrated approach highly valuable for applications where these resources are limited.

Implications for Edge ML Applications

The findings have significant implications for real-world Edge ML applications, where resource constraints pose major challenges. The integrated approach allows for the deployment of more sophisticated models on Edge devices, making it feasible for applications such as:

- **Real-time Health Monitoring:** The reduction in computational and memory demands allows for continuous monitoring and processing of health data on wearable devices, enhancing their functionality without frequent battery recharges.
- **Environmental Sensing:** Efficient data processing on constrained devices enables real-time analysis and monitoring of environmental parameters, leading to more responsive and adaptive systems.
- **Smart Home Devices:** The integration allows smart home devices to run more complex models efficiently, enhancing their ability to learn from user interactions and improve their performance over time.

Overall, the integrated approach provides a robust framework for enhancing the efficiency and performance of Edge ML models, enabling their deployment in a wider range of applications while addressing the critical constraints of computational and memory resources.

6.3 Observations and Future Enhancements

This section reflects on the core findings derived from the empirical and theoretical analyses, highlighting their significance for Edge ML and outlining promising directions for future improvements.

6.3.1 Key Insights

The integrative analysis of Data Selection, incremental learning, and efficient backpropagation techniques has yielded several key insights that significantly contribute to the advancement of Edge Machine Learning (Edge ML):

- **Enhanced Computational Efficiency:** The integration of these techniques results in substantial reductions in computational effort. Data Selection reduces the volume of data points processed, incremental learning ensures that only necessary updates are made, and sparse backpropagation minimizes the computational load during training. The cascading benefits of these methods lead to an overall computational efficiency that is critical for Edge ML applications.
- **Optimized Memory Usage:** By combining these techniques, We achieve a significant reduction in memory usage. The selective retention of data points and the adaptive nature of the incremental learning model ensure that memory resources are utilized efficiently. This optimization is particularly important for deploying machine learning models on resource-constrained devices.
- **Maintained Model Accuracy:** Despite the reductions in computational effort and memory usage, the integrated approach maintains competitive model accuracy. This balance between efficiency and performance underscores the effectiveness of the integrative approach in real-world applications where resource constraints are a major concern.
- **Robustness and Adaptability:** The adaptive mechanisms within the integrated approach enable the model to dynamically respond to changes in data and resource availability. This adaptability is crucial for Edge ML applications operating in dynamic environments, such as wearable devices, IoT applications, and smart sensors.

These insights demonstrate how the integrative approach enhances the practical applicability of Edge ML by addressing key limitations related to computational and memory constraints. The resulting improvements pave the way for more sophisticated and efficient Edge ML solutions in various domains.

6.3.2 Future Directions

Building on the findings of this study, several future research directions and improvements can be explored to further enhance the integration of Data Selection, incremental learning, and efficient backpropagation techniques:

- **Advanced Data Augmentation Techniques:** Future work could explore the integration of advanced data augmentation methods to enhance the robustness of the model further. Techniques such as synthetic data generation, adversarial training, and real-time data augmentation could be investigated to complement the existing Data Selection process.
- **Improved Adaptive Mechanisms:** The adaptivity of the model can be enhanced by incorporating more sophisticated adaptive learning algorithms. Techniques such as meta-learning, reinforcement learning, and evolutionary algorithms could provide more dynamic and context-aware adaptation strategies.
- **Real-world Deployments and Case Studies:** Conducting real-world deployments of the integrated approach across various Edge ML platforms and applications will provide valuable insights into its practical performance. Case studies in domains such as healthcare, environmental monitoring, and smart cities could highlight the benefits and challenges of the approach in real-world settings.
- **Integration with Other Complementary Techniques:** Exploring the integration of other complementary techniques, such as transfer learning, federated learning, and edge computing strategies, could further enhance the efficiency and scalability of Edge ML solutions. These techniques could be combined with the current approach to address specific challenges and optimize performance in diverse scenarios.
- **Long-term Performance Monitoring:** Implementing mechanisms for long-term performance monitoring and continual learning will ensure that Edge ML models remain effective and relevant over extended periods. Techniques for automated model updating, anomaly detection, and self-healing systems could be integrated to maintain model accuracy and efficiency continuously.

These future directions aim to build on the strengths of the current integrative approach, addressing its limitations and expanding its applicability. By pursuing these research avenues, the potential of Edge ML can be further realized, enabling more intelligent, efficient, and adaptive machine learning solutions for a wide range of applications.

Chapter 7

Conclusion and Future Work

In this chapter, key findings of the research on training neural networks in resource-constrained systems are summarized, with a focus on implications for Edge ML applications and future directions. Enhancements in computational efficiency, memory optimization, and model accuracy are discussed, alongside benefits for device autonomy, scalability, and privacy.

7.1 Summary of Key Findings

In this section, the key findings of the research on training neural networks in resource-constrained systems are summarized. The discussion focuses on enhanced computational efficiency, optimized memory usage, maintained model accuracy, robustness and adaptability, and the implications of these advancements for Edge ML applications.

7.1.1 Research Questions and Answers

The research in this thesis was structured around the following key questions, which were addressed through novel techniques and methodologies presented in the papers:

1. **Q1: How can Data Selection be optimized in Edge ML environments? Answer:** This question was addressed in **Chapter 3**, which introduced the DRIP (Drop Unimportant Data Points) algorithm. DRIP optimizes Data Selection in resource-constrained environments by prioritizing the most relevant data points for processing. Experimental results demonstrated significant improvements in data management, leading to a **39% reduction in storage use** without compromising model accuracy, making this approach highly effective for on-device training.

2. **Q2: What are effective incremental learning techniques for enhancing memory efficiency in Edge ML? Answer:** This question was explored in **Chapter 4**, which developed incremental learning techniques. These techniques enabled models to be updated incrementally, thereby reducing the need for full retraining and conserving memory. The research demonstrated that these techniques enhanced memory efficiency, allowing models to learn continuously in dynamic environments while maintaining performance across evolving datasets.

3. **Q3: How can sparse backpropagation methods be improved to increase computational efficiency in Edge ML? Answer:** In **Chapter 5**, the TinyProp algorithm was introduced and refined. This sparse backpropagation method dynamically adjusts the backpropagation ratio, reducing the computational effort required for training. The experimental results showed that TinyProp achieved a **2.9x reduction in computational effort** compared to traditional methods, while maintaining model accuracy, making it viable for use in resource-limited environments.

4. **Q4: In what ways can adaptive learning models be designed to dynamically adjust to changing data and environmental conditions in Edge ML applications? Answer:** The answer to this question was explored across **Chapter 6**, which collectively developed adaptive learning models. These models were shown to adjust dynamically to changing conditions in real-time. Techniques such as dynamic backpropagation and adaptive model updates ensured that the models remained robust and efficient even in unpredictable environments, such as environmental monitoring and healthcare applications.

By addressing these research questions, the thesis contributes significantly to advancing the field of Edge ML, providing solutions for enhanced computational efficiency, memory optimization, and model accuracy in resource-constrained environments.

7.1.2 Enhanced Computational Efficiency

The pursuit of enhanced computational efficiency in on-device training for neural networks is a cornerstone of this research. One of the primary strategies employed is sparse backpropagation. This approach significantly reduces the computational burden by selectively updating only a subset of model parameters that contribute most to the learning process. This approach not only minimizes the number of computations required but also accelerates the training process. This makes it feasible to perform complex neural network training directly on resource-constrained devices. By focusing computational resources on the most impactful parameters, sparse backpropagation ensures optimal utilization of the limited processing power available in these devices. This approach maintains the overall learning effectiveness without compromise.

Another key technique explored is incremental learning, which allows models to update continuously with new data without the need for complete retraining. This method is particularly beneficial for edge devices that operate in dynamic environments where data is constantly evolving. Incremental learning reduces the need to process and store large datasets, as the model incrementally learns from new data while retaining its performance on previously learned tasks. This continuous adaptation not only conserves computational resources but also enhances the model's ability to remain relevant and accurate over time. The efficiency gains from incremental learning are critical in ensuring that devices can perform real-time data processing and decision-making with minimal latency and energy consumption.

Furthermore, data distillation techniques have been employed to compress the training data into a more manageable size without losing significant information. By distilling the most relevant features from the original dataset, the model can be trained more quickly and with fewer resources. This approach also facilitates the use of smaller, more efficient neural network architectures that are specifically designed for the constrained computational capabilities of edge devices. The combination of these techniques—sparse backpropagation, incremental learning, and data distillation—creates a robust framework for enhancing computational efficiency. This enables sophisticated on-device training, making advanced Edge ML applications feasible even in the most resource-limited settings.

7.1.3 Optimized Memory Usage

Optimizing memory usage is a critical aspect of enabling efficient on-device training for neural networks, particularly in resource-constrained environments. One of the primary techniques utilized in this research is data distillation, which involves condensing the training data into a smaller, more informative subset. This process selectively retains the most critical data points and features, thereby reducing the overall memory footprint required for storage and processing. By working with distilled datasets, neural networks can be trained more efficiently, ensuring that memory resources are not overburdened. This approach is especially beneficial for edge devices, which typically have limited RAM and storage capacities.

Another significant contribution to optimized memory usage is the implementation of compact neural network architectures specifically designed for resource-limited environments. These architectures, such as those leveraging quantization and pruning techniques, reduce the number of parameters and the complexity of the model without sacrificing performance. Quantization involves representing weights and activations with lower precision, while pruning removes redundant or less significant parameters. Both methods contribute to a substantial reduction in the memory requirements for storing the model. The development and deployment of these lightweight models are essential for enabling real-time, on-device training and inference, thereby extending the capabilities of Edge ML applications.

Furthermore, the integration of incremental learning techniques enhances memory efficiency by allowing the model to learn continuously from new data without the need to store extensive historical data. Instead of retaining the entire dataset, the model updates its parameters incrementally, using only the most recent and relevant data. This approach not only conserves memory but also improves the model's ability to adapt to changing environments and data patterns. Additionally, memory-efficient algorithms such as Elastic Weight Consolidation (EWC) are employed to mitigate catastrophic forgetting by selectively preserving important weights from previous tasks. These combined strategies ensure that the neural networks can operate within the stringent memory constraints of edge devices, maintaining high performance and adaptability while minimizing memory usage.

7.1.4 Maintained Model Accuracy

Maintaining high model accuracy while optimizing for computational and memory efficiency is a significant challenge in on-device training for resource-constrained systems. This research employs a multifaceted approach to ensure that model accuracy is not compromised despite the stringent limitations of edge devices. Central to this strategy is the use of advanced incremental learning techniques, which allow models to adapt continuously to new data. By updating the model incrementally, only the most recent data is processed, which reduces the risk of overfitting. This approach ensures that the model remains accurate and relevant over time. This method not only conserves computational resources but also maintains the integrity of the model's predictive capabilities.

Another essential aspect of preserving model accuracy involves the application of data distillation techniques. By distilling the training data into a more concentrated and informative form, the model can be trained more effectively on a smaller dataset. This process involves selecting the most representative data points that capture the essential characteristics of the entire dataset. This ensures that the model learns the most important features without redundancy. Despite the reduced dataset size, the distilled data retains sufficient complexity to support accurate model training. This careful selection and compression of data enable the training of models that achieve high accuracy while operating within the limited memory and storage capacities of edge devices.

Moreover, the research integrates robust regularization techniques such as Elastic Weight Consolidation (EWC) to prevent catastrophic forgetting, a common issue in incremental learning scenarios. EWC adds a regularization term to the loss function, penalizing significant changes to the weights that are essential for previously learned tasks. This approach helps maintain the model's performance on old tasks while learning new ones, ensuring that accuracy is preserved across a broader range of data. Additionally, employing advanced optimization algorithms that dynamically adjust learning rates and leverage momentum further enhances model accuracy. These combined efforts result in a training framework that optimizes computational and memory resources. Additionally, it consistently maintains high model accuracy, making it suitable for a wide range of Edge ML applications.

7.1.5 Robustness and Adaptability

Ensuring robustness and adaptability in on-device training for neural networks is essential for the deployment of Edge ML applications in dynamic and unpredictable environments. This research emphasizes the development of models that can maintain their performance despite varying conditions and data distributions. One of the key strategies employed is the use of advanced incremental learning techniques, which allow models to update and adapt continuously as new data becomes available. This approach prevents the models from becoming obsolete as they are exposed to new data patterns, ensuring they remain relevant and accurate over time. Incremental learning techniques are particularly beneficial in applications such as environmental monitoring and healthcare, where data can change rapidly and unpredictably.

To further enhance robustness, the research integrates robust regularization methods such as Elastic Weight Consolidation (EWC) and dropout. EWC mitigates catastrophic forgetting by adding a regularization term to the loss function, which penalizes significant changes to critical weights learned from previous tasks. This ensures that the model retains knowledge of previously learned information while adapting to new data. Dropout, on the other hand, randomly deactivates a fraction of neurons during training, which helps prevent overfitting and encourages the model to learn more generalized features. These techniques collectively enhance the model's ability to perform consistently across a wide range of scenarios, making them robust against fluctuations and uncertainties in the data.

Adaptability is also bolstered through the use of dynamic model architectures that can be adjusted based on the complexity of the data and the available computational resources. Techniques such as model pruning and layer-wise relevance propagation allow for the dynamic adjustment of the model's architecture. This ensures that only the most critical components are retained and used during training. This flexibility enables the model to scale its complexity up or down as needed, optimizing resource usage while maintaining performance. Additionally, the implementation of adaptive learning rates and optimization algorithms that adjust in response to the model's performance further enhances adaptability. These adaptive mechanisms ensure that the model can efficiently and effectively learn from new data, maintaining high performance in diverse and changing environments.

7.1.6 Implications for Edge ML Applications

The advancements in on-device training for neural networks presented in this research have far-reaching implications for the field of Edge ML. They significantly enhance the capabilities and applications of edge devices. One of the primary benefits is the enablement of real-time processing and decision-making in various domains, such as healthcare, environmental monitoring, and smart homes. By leveraging efficient training techniques like sparse backpropagation and incremental learning, edge devices can now handle complex machine learning tasks without relying on constant connectivity to cloud servers. This independence from cloud-based resources reduces latency, enhances privacy, and ensures faster response times, making Edge ML applications more effective and user-friendly.

Moreover, the optimized memory usage and computational efficiency achieved through techniques such as data distillation and model pruning open new avenues for deploying Edge ML models on a wider range of devices. This includes those with severe resource constraints. Devices like wearables, IoT sensors, and low-power microcontrollers can now perform sophisticated data analysis and machine learning tasks locally, expanding the scope of potential applications. For instance, in remote health monitoring, wearables equipped with efficient on-device models can continuously analyze vital signs and detect anomalies. This provides timely alerts without the need for extensive data transmission. This capability not only enhances the functionality of such devices but also makes them more accessible and cost-effective.

Furthermore, the robustness and adaptability of the developed techniques ensure that Edge ML models can operate reliably in diverse and dynamic environments. This adaptability is essential for applications that require continuous learning and adaptation to new data, such as predictive maintenance in industrial settings or adaptive learning in educational technologies. The ability of Edge ML models to update incrementally and maintain accuracy over time ensures that these applications remain effective even as the underlying data changes. The implications for Edge ML are profound, as these advancements not only improve existing applications. They also pave the way for innovative solutions that were previously infeasible due to resource limitations. As a result, the research contributes to the broader adoption and impact of Edge ML technologies, driving advancements across multiple sectors.

7.2 Implications for the Edge ML Field

In this section, the implications of the research on on-device training for neural networks in Edge ML applications are explored. The discussion covers enabling advanced applications, enhancing device autonomy, improving scalability and deployment, cost efficiency, advancing research and development, implications for privacy and security, and future ecosystem development.

7.2.1 Enabling Advanced Applications

The advancements in on-device training for neural networks significantly enable the development of advanced applications that were previously constrained by the limitations of edge devices. By optimizing computational efficiency and memory usage, edge devices can now support more sophisticated machine learning models that facilitate real-time data analysis and decision-making. For example, in healthcare, wearable devices can now run complex neural networks to monitor vital signs and detect anomalies. These include irregular heartbeats or early symptoms of diseases, detected in real-time. This capability allows for timely interventions and continuous health monitoring, improving patient outcomes and reducing the need for hospital visits.

In the field of environmental monitoring, the enhanced capabilities of Edge ML enable the deployment of advanced sensor networks that can analyze environmental data locally. These sensors can detect and respond to changes in environmental conditions, such as air quality, temperature, and humidity, without the need for constant data transmission to centralized servers. This local processing capability is essential for applications in remote or resource-limited areas where connectivity may be unreliable. Additionally, the ability to perform on-device training allows these sensors to adapt to new environmental patterns over time, ensuring that the data collected remains accurate and relevant.

The advancements also pave the way for more intelligent and responsive smart home devices. With the capability to perform on-device training, smart home systems can learn from user interactions and environmental changes to provide more personalized and efficient services. For instance, smart thermostats can learn and adapt to the inhabitants' preferences and routines, optimizing energy usage while maintaining comfort. Similarly,

smart security systems can continuously update their models to recognize and respond to new types of threats or unusual activities. These advanced applications highlight the transformative potential of Edge ML, as edge devices become more capable of handling complex tasks. This leads to smarter, more responsive, and more efficient systems across various domains.

7.2.2 Enhancing Device Autonomy

The integration of advanced on-device training techniques significantly enhances the autonomy of edge devices, enabling them to operate independently without the constant need for cloud connectivity. This autonomy is essential in applications where real-time decision-making and rapid response are required. By processing and analyzing data locally, devices can reduce latency and make timely decisions, which is particularly beneficial in critical applications such as autonomous vehicles and industrial automation. In these scenarios, the ability to train and update models on-device allows for immediate adaptation to changing conditions, improving safety and operational efficiency.

Moreover, enhanced device autonomy through on-device training contributes to greater data privacy and security. As edge devices can process and store data locally, the need to transmit sensitive information to external servers is minimized. This reduction in data transmission lowers the risk of data breaches and unauthorized access. As a result, on-device training becomes a preferred approach for applications involving personal or sensitive data, such as healthcare and financial services. Patients' health data, for instance, can be analyzed and monitored directly on wearable devices, ensuring privacy and compliance with data protection regulations while still providing real-time health insights.

Additionally, autonomous edge devices equipped with on-device training capabilities can operate in remote or resource-constrained environments where connectivity is limited or unavailable. For example, environmental monitoring sensors deployed in isolated regions can continuously learn and adapt to new data patterns without relying on intermittent internet connections. This capability ensures that these devices remain functional and effective, providing valuable insights and alerts based on local data processing. The enhanced autonomy of edge devices, facilitated by efficient on-device training, expands the range of potential applications. This improvement also enhances the resilience and reliability of systems operating in challenging environments.

7.2.3 Scalability and Deployment

The techniques developed for on-device training of neural networks greatly enhance the scalability and deployment of Edge ML applications across a wide range of devices and environments. One of the primary benefits is the ability to deploy lightweight, efficient models that can operate within the stringent resource constraints of various edge devices. These devices range from low-power microcontrollers to more capable embedded systems. This scalability ensures that sophisticated machine learning capabilities are no longer confined to powerful servers but are accessible on a multitude of devices. This enables

the widespread adoption and integration of Edge ML solutions in diverse applications.

The optimized computational efficiency and memory usage of the developed techniques facilitate the deployment of Edge ML models at scale. By reducing the resource requirements for training and inference, these models can be deployed on large fleets of devices without the need for extensive hardware upgrades. This capability is particularly valuable in industries such as smart agriculture, where numerous sensors and actuators must be deployed across vast areas to monitor and respond to environmental conditions in real-time. The ability to train models on-device ensures that each device can adapt to local conditions and provide accurate, context-specific insights, enhancing the overall effectiveness of the deployment.

Furthermore, the deployment of Edge ML models is streamlined by the techniques that allow for incremental updates and continuous learning. This adaptability means that deployed models do not require frequent retraining from scratch or constant updates from centralized servers, which can be logistically challenging and costly. Instead, models can be updated incrementally as new data becomes available, ensuring they remain accurate and relevant over time. This capability significantly reduces the maintenance overhead and improves the scalability of Edge ML deployments. In applications such as predictive maintenance, where devices need to adapt to evolving operational conditions, the ability to perform on-device updates ensures that models stay current and effective. This capability provides reliable performance over extended periods.

7.2.4 Cost Efficiency

The advancements in on-device training for neural networks present significant cost efficiency benefits for the deployment and operation of Edge ML applications. By optimizing computational efficiency and memory usage, the need for expensive, high-performance hardware is reduced. This allows for the use of more affordable, resource-constrained devices, making it economically feasible to deploy machine learning capabilities on a large scale. In industries such as agriculture, healthcare, and manufacturing, where numerous devices may be required, the cost savings from using less expensive hardware can be substantial. These savings enable broader adoption and more extensive implementations of Edge ML solutions.

Additionally, the reduction in data transmission costs is a major contributor to overall cost efficiency. Traditional machine learning models often rely on sending large volumes of data to centralized servers for processing and analysis, incurring significant bandwidth and cloud storage costs. On-device training minimizes this need by processing data locally, thus reducing the amount of data that needs to be transmitted and stored externally. This not only lowers operational costs but also decreases the reliance on robust and expensive network infrastructure. For applications deployed in remote or bandwidth-limited environments, this reduction in data transmission can lead to considerable cost savings and improved reliability.

Moreover, the ability to perform incremental learning and continuous model updates on-device further enhances cost efficiency. This is achieved by reducing the maintenance and operational expenses associated with model retraining and deployment. Instead of

periodically retraining models from scratch using vast amounts of data and computational resources, models can be updated incrementally as new data becomes available. This approach reduces the frequency and intensity of retraining cycles, leading to lower computational costs and less downtime. In sectors such as predictive maintenance and smart cities, timely updates are essential for optimal performance. This capability ensures that systems remain effective without incurring high operational costs. The cumulative cost savings from these efficiencies make the deployment of advanced Edge ML applications economically viable, promoting widespread use and innovation.

7.2.5 Advancing Research and Development

The innovative techniques developed for on-device training of neural networks significantly contribute to the advancement of research and development in the field of Edge ML. This research addresses the fundamental challenges of computational efficiency and memory optimization. Consequently, it opens new avenues for exploring more complex and sophisticated machine learning models that can be deployed on resource-constrained devices. Researchers can now experiment with more advanced algorithms and architectures, pushing the boundaries of what is achievable in Edge ML. This progress not only enhances the capabilities of current applications but also paves the way for new and unforeseen uses of edge AI.

Furthermore, the ability to perform on-device training and incremental learning facilitates more dynamic and real-time experimentation. Researchers can deploy models in real-world environments and iteratively refine them based on real-time feedback and data. This approach accelerates the development cycle and enhances the practical applicability of their findings. This approach allows for a more agile R&D process, where models can be quickly tested, validated, and improved upon in situ. The continuous learning capabilities ensure that models evolve with changing data patterns, making the research outcomes more robust and adaptable to real-world conditions.

Additionally, the techniques developed in this research foster greater collaboration and innovation within the Edge ML community. By providing a framework for efficient on-device training, other researchers and developers can build upon these foundational advancements to create new tools, applications, and methodologies. The shared knowledge and technological improvements drive collective progress, as innovations in one area can be adapted and extended to others. This collaborative ecosystem accelerates the pace of R&D, leading to faster dissemination of breakthroughs and a more vibrant and productive field. The cumulative impact of these advancements ensures that Edge ML remains at the forefront of AI research, driving forward the capabilities and applications of edge intelligence.

7.2.6 Implications for Privacy and Security

The advancements in on-device training for neural networks carry significant implications for privacy and security in Edge ML applications. By enabling data processing and analysis to occur directly on edge devices, the need to transmit sensitive data to centralized servers is minimized. This local processing capability reduces the risk of data breaches and unauthorized access. Personal or sensitive information remains on the device and is not exposed to potential vulnerabilities during transmission. In applications such as healthcare and finance, where data privacy is paramount, on-device training ensures compliance with stringent data protection regulations, enhancing user trust and system security.

Furthermore, on-device training supports the implementation of robust security protocols tailored to the specific needs of individual devices. Since data does not need to be shared externally, edge devices can employ customized encryption and access control measures that are optimized for their specific operating environments. This level of granularity in security management is difficult to achieve with centralized systems, where uniform security policies must be applied across diverse and distributed data sources. The ability to enforce device-specific security measures enhances the overall resilience of the system against targeted attacks and exploits. This provides a more secure and reliable platform for deploying Edge ML applications.

Additionally, the inherent adaptability of on-device training allows for continuous monitoring and updating of security protocols in response to emerging threats. As models are trained and updated incrementally, security algorithms can also be refined and adapted to address new vulnerabilities and attack vectors. This dynamic approach ensures that security measures remain effective over time, protecting against evolving cyber threats. In highly dynamic environments like smart cities or industrial IoT systems, devices must rapidly adapt and enhance security protocols. This capability is essential for maintaining the system's integrity and reliability. Overall, the implications for privacy and security underscore the critical role of on-device training in building trustworthy and resilient Edge ML applications.

7.2.7 Future Ecosystem Development

The advancements in on-device training for neural networks are poised to significantly influence the future development of the Edge ML ecosystem. By overcoming key challenges related to computational efficiency and memory usage, these techniques enable the creation of more powerful and efficient edge devices. This progress fosters an ecosystem where diverse applications can thrive, from smart homes and wearable technology to industrial automation and environmental monitoring. As more devices become capable of performing complex machine learning tasks independently, the ecosystem will expand, integrating more sophisticated functionalities and offering richer, more adaptive user experiences.

The growth of this ecosystem will be driven by collaboration and standardization within the Edge ML community. The development of common frameworks and protocols

for on-device training will facilitate interoperability among devices and applications, allowing different systems to work seamlessly together. This standardization will also lower the barriers to entry for new developers and researchers, encouraging innovation and accelerating the adoption of Edge ML technologies. As these standards evolve, they will support the integration of diverse technologies, such as IoT, 5G, and blockchain, further enhancing the capabilities and security of the ecosystem.

Moreover, the future ecosystem development will be characterized by continuous innovation and refinement of on-device training techniques. As new challenges and opportunities arise, ongoing research will focus on enhancing the robustness, adaptability, and efficiency of Edge ML models. This iterative process will ensure that the ecosystem remains dynamic and responsive to changing technological and market demands. In particular, advancements in areas like federated learning and edge AI will drive the next generation of Edge ML applications, enabling more intelligent, autonomous, and secure systems. These developments will cumulatively result in a vibrant and robust Edge ML ecosystem. This ecosystem can support a wide range of advanced applications, driving forward the frontier of edge intelligence.

7.3 Future Directions

In this section, the future directions for enhancing on-device training of neural networks in Edge ML applications are explored. Topics include advanced data augmentation techniques, improved adaptive mechanisms, real-world deployments and case studies, integration with other complementary techniques, and long-term performance monitoring.

7.3.1 Advanced Data Augmentation Techniques

Advanced data augmentation techniques play a essential role in enhancing the performance and robustness of on-device training for neural networks in resource-constrained environments. Data augmentation involves generating additional training examples by transforming existing data, which helps improve model generalization and prevent overfitting. Techniques such as geometric transformations (e.g., rotations, translations, and scaling), color adjustments, and noise injection are commonly used to create diverse and representative training datasets. These methods are particularly valuable in Edge ML applications, where collecting large amounts of labeled data can be challenging. By synthetically expanding the dataset, advanced data augmentation ensures that models trained on-device can achieve high accuracy and robustness even with limited original data.

Recent advancements in data augmentation have introduced more sophisticated techniques such as adversarial training and generative adversarial networks (GANs). Adversarial training involves generating challenging examples that are specifically designed to fool the model, thus making it more robust against adversarial attacks. GANs, on the other hand, can create highly realistic synthetic data that closely mimics the distribution of the original dataset. These advanced techniques not only enhance the diversity of the

training data but also improve the model's ability to handle edge cases and rare scenarios. For on-device training, integrating these methods can significantly boost the model's performance and resilience, ensuring that it performs well in real-world conditions where data variability is high.

Furthermore, the implementation of real-time data augmentation techniques directly on edge devices can provide additional benefits. Techniques such as Differentiable Data Augmentation (DDA) and AutoAugment use reinforcement learning to automatically discover the best augmentation policies for a given task, optimizing the training process dynamically. By performing data augmentation in real-time during the training phase, edge devices can continuously adapt to new data patterns and environmental changes. This approach helps maintain high levels of accuracy and robustness. This capability is particularly important for applications like autonomous vehicles and smart surveillance systems, where the operational environment can change rapidly and unpredictably. Overall, advanced data augmentation techniques are essential for maximizing the potential of on-device training, ensuring that Edge ML models are both effective and resilient in diverse and dynamic contexts.

7.3.2 Improved Adaptive Mechanisms

Improved adaptive mechanisms are essential for enhancing the efficiency and effectiveness of on-device training in resource-constrained environments. Adaptive mechanisms adjust the learning process dynamically based on the data and the current state of the model, ensuring optimal performance with minimal computational overhead. Techniques such as adaptive learning rates, where the step size for updating model parameters changes based on the gradient, help in converging faster and avoiding local minima. By implementing these mechanisms, edge devices can maintain high accuracy and efficiency even as the complexity of tasks and data variability increases.

One significant advancement in this area is the development of meta-learning algorithms, also known as "learning to learn" methods. Meta-learning enables models to adapt quickly to new tasks by leveraging prior knowledge gained from previous tasks. This is particularly beneficial in Edge ML applications where the model must adapt to new data continuously. Algorithms such as Model-Agnostic Meta-Learning (MAML) and Reptile allow the model to optimize its learning process. This makes the model more flexible and capable of handling a wide range of tasks with minimal retraining. These methods enhance the model's adaptability, ensuring that it can efficiently learn new patterns and behaviors, which is essential for applications such as personalized healthcare and adaptive robotics.

Moreover, the integration of reinforcement learning techniques into on-device training frameworks provides another layer of adaptability. Reinforcement learning enables models to learn optimal behaviors through trial and error, guided by a reward signal. Techniques like Proximal Policy Optimization (PPO) and Deep Q-Networks (DQN) can be used to develop adaptive mechanisms. These mechanisms fine-tune model parameters in real-time based on the performance feedback from the environment. This approach is particularly useful for dynamic and interactive applications, such as autonomous navi-

gation and real-time decision-making systems. By leveraging improved adaptive mechanisms, Edge ML applications can achieve a higher degree of intelligence and autonomy, ensuring robust performance across a wide range of scenarios and operating conditions.

7.3.3 Real-World Deployments and Case Studies

Real-world deployments and case studies are critical for validating the effectiveness and practicality of on-device training techniques developed for Edge ML applications. These deployments provide valuable insights into how these techniques perform in diverse, real-world environments and under various operational constraints. For instance, deploying Edge ML models in smart city infrastructures, such as traffic monitoring systems, can demonstrate the models' ability to process and analyze data in real-time. These models can adapt to changing conditions and provide actionable insights to improve urban mobility and safety. These case studies highlight the potential benefits and challenges of integrating Edge ML solutions into complex, real-world systems.

In the healthcare sector, real-world deployments of on-device trained models in wearable devices and remote monitoring systems offer a glimpse into the transformative potential of Edge ML. For example, wearables equipped with Edge ML models can continuously monitor patients' vital signs and detect anomalies, providing early warnings for potential health issues. Case studies from such deployments can illustrate the models' accuracy, reliability, and impact on patient outcomes. These studies showcase how on-device training can enhance personalized healthcare and reduce the burden on centralized medical facilities. These deployments also reveal important considerations for data privacy and security, as sensitive health data is processed locally on the devices.

Furthermore, case studies in industrial settings, such as predictive maintenance of machinery, underscore the practical advantages of on-device training. By deploying Edge ML models directly on industrial sensors and equipment, companies can monitor the health of their machinery in real-time, predict failures, and schedule maintenance proactively. These real-world applications demonstrate the models' ability to operate under harsh conditions, handle noisy data, and provide timely insights that prevent costly downtime and improve operational efficiency. Documenting these case studies validates the technical feasibility of on-device training. It also provides a roadmap for future implementations, highlighting best practices, potential pitfalls, and areas for further improvement.

7.3.4 Integration with Other Complementary Techniques

Integrating on-device training with other complementary techniques can significantly enhance the capabilities and performance of Edge ML applications. One such complementary technique is federated learning, which allows models to be trained across multiple decentralized devices while keeping the data localized. By combining on-device training with federated learning, it is possible to aggregate insights from numerous edge devices without compromising data privacy. This approach enables collaborative learning across a distributed network of devices, enhancing the overall model accuracy and

robustness. For example, in a healthcare setting, wearable devices could collaboratively learn from the data of numerous patients to improve diagnostic models. This approach ensures that sensitive health data remains on each patient's device.

Another complementary technique is transfer learning, which involves leveraging pre-trained models on similar tasks to accelerate the training process for new tasks. Integrating transfer learning with on-device training allows edge devices to benefit from the knowledge encoded in large-scale models trained on extensive datasets. This integration can be particularly effective in scenarios where labeled data is scarce or expensive to obtain. For instance, a pre-trained model on general object recognition can be fine-tuned on-device to recognize specific objects relevant to a particular application. This could involve identifying machinery parts in an industrial setting. This approach reduces the computational and data requirements for on-device training, making it more feasible to deploy sophisticated models on resource-constrained devices.

Furthermore, the integration of edge computing and on-device training can create a more efficient and resilient computational architecture. Edge computing involves processing data closer to its source rather than relying on centralized cloud servers. By combining on-device training with edge computing, data can be processed and analyzed locally, reducing latency and bandwidth usage. This integration also allows for more complex and computationally intensive tasks to be offloaded to nearby edge servers when necessary, balancing the computational load and optimizing resource utilization. In smart city applications, traffic data from various sensors can be processed locally to manage traffic flow in real-time. Additionally, more extensive data analysis can be performed on edge servers to identify long-term trends and patterns. The synergy between these complementary techniques and on-device training enhances the scalability, efficiency, and adaptability of Edge ML applications across various domains.

7.3.5 Long-Term Performance Monitoring

Long-term performance monitoring is essential for ensuring the sustained effectiveness and reliability of on-device trained neural networks in Edge ML applications. Continuous monitoring allows for the detection of performance degradation over time. This degradation can be caused by various factors such as changes in data distribution, environmental conditions, or hardware wear and tear. By implementing robust performance monitoring frameworks, it is possible to track key metrics such as accuracy, latency, and resource utilization. This tracking provides insights into the model's behavior in real-world conditions. This ongoing assessment is essential for maintaining the relevance and accuracy of the deployed models, ensuring they continue to meet the application's requirements.

To effectively monitor long-term performance, it is important to incorporate automated mechanisms that can detect and respond to deviations in model behavior. Techniques such as anomaly detection can be used to identify unusual patterns or significant drops in performance, triggering alerts or automated retraining processes. For example, in predictive maintenance applications, continuous monitoring of the model's predictions against actual equipment performance can highlight discrepancies that indicate model

drift. Automated retraining using recent data can then be initiated to update the model, ensuring it remains accurate and reliable. These proactive measures help prevent failures and maintain high levels of operational efficiency.

Moreover, long-term performance monitoring can provide valuable feedback for improving future iterations of Edge ML models. Data collected from monitoring can be analyzed to understand the factors contributing to performance changes, guiding enhancements in model architecture, training techniques, and deployment strategies. This iterative improvement process is critical for advancing the state of the art in Edge ML, as it enables the development of more robust and adaptive models. Additionally, insights gained from long-term monitoring can inform the design of new applications, ensuring they are built with considerations for real-world variability and longevity. Overall, long-term performance monitoring not only sustains the effectiveness of current Edge ML deployments but also drives continuous innovation and improvement in the field.

Appendix A

Dataset Descriptions

This section provides detailed descriptions of the datasets used in this work. Each dataset is labeled for reference throughout the main text.

A.0.1 MNIST

The MNIST dataset (Modified National Institute of Standards and Technology database) is a collection of grayscale images of handwritten digits, commonly used for evaluating image classification models. It consists of:

- **Training Set:** 60,000 examples.
- **Test Set:** 10,000 examples.
- **Image Details:** Each image is 28x28 pixels, normalized and centered using the center of mass of the pixels.

The dataset is derived from the NIST Special Database 3 and 1, and is renowned for its simplicity and accessibility. [106]

A.0.2 CIFAR-10

The CIFAR-10 dataset is a widely-used benchmark for image classification tasks. It comprises:

- **Total Images:** 60,000 32x32 color images.
- **Classes:** 10 categories, including airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks.
- **Split:** 50,000 training images and 10,000 test images.

Each image represents a single object and is photorealistic. [76]

A.0.3 Plant Disease (PD)

The Plant Disease dataset is designed for classifying crop health and comprises:

- **Total Images:** Approximately 87,000 RGB images.
- **Classes:** 38 categories representing healthy and diseased leaves.
- **Split:** 80% training and 20% validation, with an additional curated test set.

This dataset highlights the complexity of agricultural applications. [107]

A.0.4 Speech Commands (SC)

The Speech Commands dataset is tailored for training and evaluating keyword spotting systems. It includes:

- **Audio Clips:** 65,000 one-second-long audio samples.
- **Words:** 30 distinct spoken keywords by various speakers.

This dataset challenges models with nuances in audio signals and serves as a benchmark for speech recognition systems. [77]

A.0.5 CORe50

CORe50 is specifically designed for continual object recognition tasks. It consists of:

- **Objects:** 50 domestic objects divided into 10 categories.
- **Task Variants:** Classification can be at the object or category level.
- **Setup:** Objects are handheld, with varying poses and backgrounds, adding to the challenge.

This dataset simulates realistic scenarios for robotic and continual learning applications. [109]

A.0.6 ESC-50

The ESC-50 dataset is designed for environmental sound classification and includes:

- **Recordings:** 2,000 five-second-long audio clips.
- **Classes:** 50 semantically distinct categories, loosely grouped into five major categories.

It provides a benchmark for audio classification tasks. [110]

A.0.7 Human Activity Recognition (HAR)

The HAR dataset captures daily activities using wearable sensors. It comprises:

- **Subjects:** 30 individuals performing various activities.
- **Sensors:** Accelerometer and gyroscope data collected via waist-mounted smartphones.
- **Data Type:** Multi-dimensional time-series sensor data.

This dataset tests activity recognition models in dynamic, real-world conditions. [111]

A.0.8 DCASE2020 Challenge Task 1

Derived from the DCASE2020 Challenge, this dataset focuses on sound-based anomaly detection in machines. For this study:

- **Machine Types:** Six distinct machine types, with a focus on the slide rail.
- **Task:** Binary classification of normal vs. anomalous operational sounds.

This dataset is crucial for evaluating anomaly detection systems in industrial applications. [116]

A.0.9 Oxford Flowers 102

The Oxford Flowers 102 dataset presents a fine-grained classification challenge, capturing diverse flower species. It includes:

- **Total Images:** 8,189 RGB images.
- **Classes:** 102 flower categories, each representing a unique species.
- **Split:** Pre-defined splits of training, validation, and test sets.

This dataset is known for its detailed categorization and is a benchmark for fine-grained visual classification tasks. [117]

A.0.10 Food-101

The Food-101 dataset simulates real-world scenarios with noisy and imbalanced data, making it a robust benchmark for image recognition. It includes:

- **Total Images:** 101,000 RGB images, with 1,000 images per class.
- **Classes:** 101 categories representing diverse food dishes.
- **Challenges:** Contains noisy labels and real-world image variations.

This dataset emphasizes the importance of handling noise and imbalance in practical applications. [118]

Appendix B

Supporting Material on Neural Network Training

B.1 Forward and Backward Propagation

Deep Neural Networks (DNNs) rely on two fundamental processes: forward propagation and backward propagation. Forward propagation computes the output of the network for a given input, while backward propagation adjusts the network's weights based on the error of its predictions. These processes are the core mechanisms for training DNNs. The following sections detail these processes mathematically and conceptually. For further reading, see [119].

B.1.1 Forward Propagation

Forward propagation involves passing input data through the network layer by layer to compute the final output. For a given layer l, the operations are described as follows:

$$a^{l+1} = f\left(z^l\right) = f\left(W^l a^l + b^l\right) \tag{B.1}$$

where:

- W^l: Weight matrix for layer l,
- b^l: Bias vector for layer l,
- z^l: Weighted sum of inputs for layer l,
- a^l: Activation (output) of layer l,
- f: Non-linear activation function applied element-wise.

This process continues sequentially through the layers until the final output of the network is computed. Forward propagation is computationally efficient and forms the basis for generating predictions.

B.1.2 Backward Propagation

Backward propagation is used to update the weights and biases of the network based on the error (or loss) computed during forward propagation. This process ensures that the network improves its predictions over successive iterations.

The loss function $\mathcal{L}$ quantifies the error between the predicted output a^L and the ground truth y:

$$\mathcal{L}(a^L, y) \tag{B.2}$$

Gradient Computation

The gradients of the loss with respect to the outputs of the final layer z^L are computed as:

$$\delta_z^L = \left(\frac{\partial \mathcal{L}}{\partial a^L} \frac{\partial a^L}{\partial z^L} \right)^T = f'\left(z^L\right) \odot \nabla_{a^L} \mathcal{L} \tag{B.3}$$

Here:

- f': Derivative of the activation function,
- $\nabla_{a^L} \mathcal{L}$: Gradient of the loss with respect to the final layer's activation,
- $\odot$: Hadamard (element-wise) product.

Weight and Bias Updates

Using the chain rule, the gradients are propagated backward through the network to compute updates for each layer. For layer l, the updates are:

$$\delta_z^l = \delta_a^l \odot f'\left(z^l\right) \tag{B.4}$$

$$W^l \leftarrow W^l - \eta \cdot \delta_z^l \cdot a^{l-1T} \tag{B.5}$$

$$b^l \leftarrow b^l - \eta \cdot \delta_z^l \tag{B.6}$$

where:

- η: Learning rate,
- δ_a^l: Gradient of the loss with respect to the activation of layer l,
- δ_z^l: Gradient of the loss with respect to the pre-activation of layer l.

Backward propagation ensures that the weights and biases are updated to minimize the loss, thereby improving the network's predictions over time.

B.2 Conclusion

Forward and backward propagation are the cornerstone of training DNNs. While forward propagation computes outputs efficiently, backward propagation iteratively adjusts the weights and biases to minimize error. These processes together enable the dynamic learning capabilities of modern neural networks.

B.3 Evaluation Metrics

In this appendix, we provide detailed explanations of the evaluation metrics used in this work, including accuracy, precision, recall, and F1-score. These metrics are widely used to evaluate machine learning models and are defined as follows:

B.3.1 Accuracy

Accuracy is defined as the proportion of correct predictions to the total number of predictions:

$$A = \frac{C_{\text{correct}}}{N_{\text{total}}} \tag{B.7}$$

Bibliography

[1] M. Rüb and A. Sikora. A practical view on training neural networks in the edge. *IFAC-PapersOnLine*, 55(4):272–279, 2022. doi: 10.1016/j.ifacol.2022.06.045.

[2] S. Li, C. Tian, K. Tam, R. Ma, and L. Li. Breaking on-device training memory wall: A systematic survey. *arXiv preprint arXiv:2306.10388*, 2023.

[3] H. Cai, C. Gan, L. Zhu, and S. Han. Tinytl: Reduce memory, not parameters for efficient on-device learning. In *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, Vancouver, BC, Canada, Dec. 6–12 2020. doi: 10.5555/3495724.3496671.

[4] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, Oct. 1950. ISSN 0026-4423. doi: 10.1093/mind/LIX.236.433.

[5] S. Russell and P. Norvig. *Künstliche Intelligenz: Ein moderner Ansatz*. I - Informatik. Pearson Studium, München, Germany, 2nd ed. edition, 2007. ISBN 3827370892.

[6] P. Sharma, M. Rüb, D. Gaida, H. Lutz, and A. Sikora. Deep learning in resource and data constrained edge computing systems. In B. Beyerer, editor, *Machine Learning for Cyber Physical Systems*, volume 13 of *Technologien für die intelligente Automation*, pages 43–51. Springer Berlin Heidelberg, 2021. ISBN 978-3-662-62745-7.

[7] V. Rajapakse, I. Karunanayake, and N. Ahmed. Intelligence at the extreme edge: A survey on reformable tinyml. *ACM Computing Surveys*, 55(13s):1–30, 2023. ISSN 0360-0300. doi: 10.1145/3583683.

[8] P. Warden and D. Situnayake. *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O'Reilly Media, Beijing, China, 2019. ISBN 9781492051992.

[9] H. Doyu, R. Morabito, and M. Brachmann. A tinymlaas ecosystem for machine learning in iot: Overview and research challenges. In *2021 International Symposium on VLSI Design, Automation and Test (VLSI-DAT 2021) – Proceedings*. IEEE, 2021. doi: 10.1109/VLSI-DAT52063.2021.9427352.

[10] Y. Zhang, N. Suda, L. Lai, and V. Chandra. Hello edge: Keyword spotting on microcontrollers. *arXiv preprint arXiv:1711.07128*, 2017.

[11] M. de Prado, M. Rusci, A. Capotondi, R. Donze, L. Benini, and N. Pazos. Robustifying the deployment of tinyml models for autonomous mini-vehicles. *Sensors*, 21(4):1339, 2021. doi: 10.3390/s21041339.

[12] L. Dutta and S. Bharali. Tinyml meets iot: A comprehensive survey. *Internet of Things*, 16:100461, 2021. ISSN 2542-6605. doi: 10.1016/j.iot.2021.100461.

[13] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag. What is the state of neural network pruning? In *Proceedings of Machine Learning and Systems (MLSys)*, volume 2, pages 129–146, 2020.

[14] W. Park, D. Kim, Y. Lu, and M. Cho. Relational knowledge distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3967–3976, 2019. doi: 10.1109/CVPR.2019.00409.

[15] Y. Cai, Z. Yao, Z. Dong, A. Gholami, M. W. Mahoney, and K. Keutzer. Zeroq: A novel zero shot quantization framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13169–13178, June 2020. doi: 10.1109/CVPR42600.2020.01318.

[16] R. David, J. Duke, A. Jain, V. Janapa Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, T. Wang, P. Warden, and R. Rhodes. Tensorflow lite micro: Embedded machine learning on tinyml systems. In *Proceedings of the 3rd Machine Learning and Systems (MLSys) Conference*, pages 800–811, 2021. doi: 10.48550/arXiv.2010.08678.

[17] S. Disabato and M. Roveri. Incremental on-device tiny machine learning. In *Proceedings of the 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things (CAIL IoT'20)*, pages 7–13. Association for Computing Machinery, 2020. ISBN 978-1-4503-8134-5. doi: 10.1145/3417313.3429378.

[18] R. Sanchez-Iborra and A. F. Skarmeta. Tinyml-enabled frugal smart objects: Challenges and opportunities. *IEEE Circuits and Systems Magazine*, 20(3):4–18, Jul.–Sep. 2020. doi: 10.1109/MCAS.2020.3005467.

[19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS 2019)*, volume 32, pages 8024–8035, Vancouver, Canada, Dec. 8–14 2019. Curran Associates, Inc.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay. Scikit-learn: Machine learning in python. In *Journal of Machine Learning Research*, volume 12, pages 2825–2830, Nov. 2011.

[21] C. Banbury, C. Zhou, I. Fedorov, R. Matas, U. Thakker, D. Gope, V. J. Reddi, M. Mattina, and P. N. Whatmough. Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers. In *Proceedings of the 3rd Machine Learning and Systems (MLSys)2021*, pages 517–532, Virtual (online), Apr. 2021.

[22] Edge Impulse Team. Projects — edge impulse blog, Jul 2022. URL https://www.edgeimpulse.com/blog/category/projects/. Accessed 2025-03-04.

[23] Cartesiam. Cartesiam, nanoedge ai library, Dec 2021. URL https://cartesiam.ai/. Accessed 2025-03-04.

[24] C. Banbury, V. J. Reddi, P. Torelli, J. Holleman, N. Jeffries, C. Király, P. Montino, D. Kanter, S. Ahmed, D. Pau, U. Thakker, A. Torrini, P. Warden, J. Cordaro, G. Di Guglielmo, J. Duarte, S. Gibellini, V. Parekh, H. Tran, N. Tran, W. X. Niu, and X. Xuesong. Mlperf tiny benchmark. In *Proceedings of the 35th Conference on Neural Information Processing Systems (NeurIPS) – Track on Datasets Benchmarks*, Dec. 2021. doi: 2106.07597.

[25] Microsoft. Microsoft/edgeml: Code for machine learning algorithms for edge devices developed at microsoft research india, Feb. 2022. URL https://github.com/Microsoft/EdgeML. Version 0.4 (see GitHub history).

[26] C. Lalau-Keraly, G. Daniel, J. Lee, and D. Schwartz. Peel-and-stick sensors powered by directed rf energy. In *Proceedings of the ASME 2017 International Technical Conference and Exhibition on Packaging and Integration of Electronic and Photonic Microsystems (InterPACK & ISPS)*, volume 58097, page V002T02A001, Cleveland, OH, USA, Jun. 2017. American Society of Mechanical Engineers (ASME). doi: 10.1115/IPACK2017-74150.

[27] T. Wu, J.-M. Redouté, and M. R. Yuce. A wireless implantable sensor design with subcutaneous energy harvesting for long-term iot healthcare applications. *IEEE Access*, 6:35801–35808, Jul. 2018. doi: 10.1109/ACCESS.2018.2851940.

[28] R. Zhu, M. Yang, and Q. Wang. Shufflefl: Addressing heterogeneity in multi-device federated learning. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)*, 8(2):85:1–85:34, May 2024. doi: 10.1145/3659621.

[29] S. Sankar and P. Srinivasan. Composite metric based energy efficient routing protocol for internet of things. *International Journal of Intelligent Engineering and Systems*, 10(5):278–286, Oct. 2017. doi: 10.22266/ijies2017.1031.30.

[30] Z. Zhang, L. Shu, C. Zhu, and M. Mukherjee. A short review on sleep scheduling mechanism in wireless sensor networks. In *International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness (QShine 2017)*, Lecture Notes of the Institute for Computer Sciences, Social Informatics

and Telecommunications Engineering (LNICST), Vol. 234, pages 66–70. Springer, Cham, Switzerland, Mar. 2017. doi: 10.1007/978-3-319-78078-8-7.

[31] Y. Huang and H. Haddadi. Poster: Towards battery-free machine learning inference and model personalization on mcus. In *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services (MobiSys '23)*, pages 571–572, Helsinki, Finland, Jun. 2023. Association for Computing Machinery. doi: 10.1145/3581791.3597371.

[32] N. Schizas, A. Karras, C. Karras, and S. Sioutas. Tinyml for ultra-low power AI and large-scale IoT deployments: A systematic review. *Future Internet*, 14(12): 363, Dec. 2022. ISSN 1999-5903. doi: 10.3390/fi14120363.

[33] V. J. Reddi, B. Plancher, S. Kennedy, L. Moroney, P. Warden, A. Agarwal, C. Banbury, M. Banzi, M. Bennett, B. Brown, S. Chitlangia, R. Ghosal, S. Grafman, R. Jaeger, S. Krishnan, M. Lam, D. Leiker, C. Mann, M. Mazumder, D. Pajak, D. Ramaprasad, J. E. Smith, M. Stewart, and D. Tingley. Widening access to applied machine learning with tinyml. *arXiv preprint*, arXiv:2106.04008, Jun. 2021.

[34] H. Ren, D. Anicic, and T. A. Runkler. Tinyol: Tinyml with online-learning on microcontrollers. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, Jul. 2021. doi: 10.1109/IJCNN52387.2021.9533927.

[35] L. Ravaglia, M. Rusci, D. Nadalini, A. Capotondi, F. Conti, and L. Benini. A tinyml platform for on-device continual learning with quantized latent replays. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 11(4): 789–802, Dec. 2021. ISSN 2156-3357. doi: 10.1109/JETCAS.2021.3121554.

[36] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, C. Gan, and S. Han. On-device training under 256 kb memory. *Proceedings of the 36th Conference on Neural Information Processing Systems (NeurIPS 2022)*, Nov. 26 2022. doi: 10.5555/3600270.3601937.

[37] Zhiyuan W., Sheng S., Yuwei W., Min L., Bo G., Tianliu H., and Wen W. Privacy-enhanced training-as-a-service for on-device intelligence: Concept, architectural scheme, and open problems. *arXiv preprint 2404.10255*, 2024.

[38] G. B. Hacene, X. Bellekens, and I. Lamprinos. Incremental learning on chip. *Proceedings of the 2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, Nov. 2017. doi: 10.1109/GlobalSIP.2017.8309068.

[39] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):1–37, Mar. 2014. ISSN 0360-0300. doi: 10.1145/2523813.

[40] M. Rüb, A. Sikora, and D. Mueller-Gritschneder. Advancing on-device neural network training with tinypropv2: Dynamic, sparse, and efficient backpropagation. In *Proceedings of the 2024 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, Jul. 2024. doi: 10.1109/IJCNN60899.2024.10650122.

[41] M. Rüb, D. Konegen, A. Sikora, and D. Mueller-Gritschneder. Drip: Drop unimportant data points – enhancing machine learning efficiency with grad-cam based real-time data prioritization for on-device training. In *2025 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2025. doi: 10.1109/IJCNN64981.2025.11228149.

[42] M. Rüb, P. Tuchel, A. Sikora, and D. Mueller-Gritschneder. A continual and incremental learning approach for tinyml on-device training using dataset distillation and model size adaption. In *2024 IEEE 7th International Conference on Industrial Cyber-Physical Systems (ICPS)*, pages 1–8, 2024. doi: 10.1109/ICPS59941.2024.10639989.

[43] Google Coral Team. Edge tpu: Google's asic for edge ai. Online documentation, Jan. 2020. URL https://coral.ai/docs/edgetpu/. Accessed: Mar. 1, 2025.

[44] NVIDIA Corporation. Nvidia jetson: Ai at the edge, 2023. URL https://developer.nvidia.com/embedded/jetson. Accessed: Mar. 1, 2025.

[45] A. M. Kist and Y. Sun. Deep learning on edge tpus. *arXiv preprint 10.48550/arXiv.2108.13732*, Aug. 2021.

[46] H. Ren, D. Anicic, and T. A. Runkler. Tinyreptile: Tinyml with federated meta-learning. *Proceedings of the 2023 International Joint Conference on Neural Networks (IJCNN)*, Jun. 2023. doi: 10.1109/ijcnn54540.2023.10191845.

[47] X. Yang, H. Yu, X. Gao, H. Wang, J. Zhang, and T. Li. Federated continual learning via knowledge fusion: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 36(8):3832–3850, August 2024. ISSN 2326-3865. doi: 10.1109/tkde.2024.3363240.

[48] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[49] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50× fewer parameters and 0.5 mb model size. arXiv preprint 10.48550/arXiv.1602.07360, Feb. 2016.

[50] M. Tan and Q. V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *Proc. 36th Int. Conf. on Machine Learning (ICML)*, page 6105–6114, Jun. 9–15 2019. doi: 10.5555/3294771.3294980.

[51] R. D. Nathoo, M. Kegler, and M. Stamenovic. Two-step knowledge distillation for tiny speech enhancement. In *Proc. 2024 IEEE Int. Conf. Acoust., Speech and Signal Processing (ICASSP)*, pages 10141–10145, Seoul, Korea, Apr. 2024. doi: 10.1109/ICASSP48485.2024.10446796.

[52] H. Yuan, J. Cheng, Y. Wu, and Z. Zeng. Low-res mobilenet: An efficient lightweight network for low-resolution image classification in resource-constrained scenarios. *Multimedia Tools and Applications*, 81, Apr. 2022. doi: 10.1007/s11042-022-13157-8.

[53] Z. Li, A. Samanta, Y. Li, A. Soltoggio, H. Kim, and C. Liu. R^3: On-device real-time deep reinforcement learning for autonomous robotics. *Proc. 2023 IEEE Real-Time Systems Symposium (RTSS)*, page 131–144, Dec. 5–8 2023. doi: 10.1109/RTSS59052.2023.00021.

[54] N. Andalib and M. Selimi. Exploring local and cloud-based training use cases for embedded machine learning. *Proceedings of the 13th Mediterranean Conference on Embedded Computing (MECO)*, Jun. 11–14 2024. doi: 10.1109/MECO62516.2024.10577837.

[55] M. A. Hoque and C. Davidson. Design and implementation of an iot-based smart home security system. *International Journal of Networked and Distributed Computing*, 7(2):85–92, Apr. 2019. ISSN 2211-7946. doi: 10.2991/ijndc.k.190326.004.

[56] T. Choudhary, V. K. Mishra, A. Goswami, and S. Jagannathan. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review*, 53 (7):5113–5155, Oct. 2020. doi: 10.1007/s10462-020-09816-7.

[57] A. G. Accettola and M. Merenda. Dataset distillation as an enabling technique for on-device training in tinyml for iot: An rfid use case. In *Proc. 2023 8th Int. Conf. Smart and Sustainable Technologies (SpliTech)*, pages 1–4, Split / Bol, Croatia, Jun. 20–23 2023. doi: 10.23919/SpliTech58164.2023.10193138.

[58] A. Brecko, E. Kajati, J. Koziorek, and I. Zolotova. Federated learning for edge computing: A survey. *Applied Sciences*, 12(18):9124, Sept. 2022. ISSN 2076-3417. doi: 10.3390/app12189124.

[59] D. Li, Z. Wang, Y. Chen, R. Jiang, W. Ding, and M. Okumura. A survey on deep active learning: Recent advances and new frontiers. May 2024. doi: 10.48550/arXiv.2405.00334. arXiv preprint.

[60] C. Guo, B. Zhao, and Y. Bai. Deepcore: A comprehensive library for coreset selection in deep learning. volume abs/2204.08499, Apr. 2022. doi: 10.48550/arXiv.2204.08499. arXiv preprint.

[61] R. R. Selvaraju, M. Cogswell, A. Das, Ramakrishna Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2):336–359, October 2019. ISSN 1573-1405. doi: 10.1007/s11263-019-01228-7.

[62] E. Štrumbelj and I. Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowledge and Information Systems*, 41 (3):647–665, Aug. 2014. doi: 10.1007/s10115-013-0679-x.

[63] R. Jia, D. Dao, B. Wang, F. A. Hubis, N. Hynes, N. M. Gürel, B. Li, C. Zhang, D. Song, and C. J. Spanos. Towards efficient data valuation based on the shapley value. 89:1167–1176, Apr. 16–18 2019. doi: 10.5555/3305381.3305525.

[64] R. H. L. Sim, X. Xu, and B. K. H. Low. Data valuation in machine learning: "ingredients", strategies, and open challenges. pages 5607–5614, Vienna, Austria, Jul. 2022. International Joint Conferences on Artificial Intelligence Organization. doi: 10.24963/ijcai.2022/782.

[65] J. T. Wang and R. Jia. Data banzhaf: A robust data valuation framework for machine learning. *arXiv preprint 10.48550/arXiv.2205.15466*, May 2022.

[66] A. Xu, Z. Zheng, F. Wu, and G. Chen. Online data valuation and pricing for machine learning tasks in mobile health. In *Proc. 41st IEEE Int. Conf. on Computer Communications (INFOCOM)*, pages 850–859, Las Vegas, NV, USA, May 2–5 2022. IEEE. doi: 10.1109/INFOCOM48880.2022.9796669.

[67] Y. Ding, J. Liu, X. Xu, M. Huang, J. Zhuang, J. Xiong, and Y. Shi. Uncertainty-aware training of neural networks for selective medical image segmentation. pages 156–173, Montréal, QC, Canada, Jul. 6–8 2020. PMLR.

[68] D. S. Hochbaum and D. B. Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, May 1985. doi: 10.1287/moor.10.2.180.

[69] A. Yamani, A. Alyami, H. Luqman, B. Ghanem, and S. Giancola. Active learning for single-stage object detection in uav images. pages 1849–1858, Waikoloa, HI, USA, Jan. 4–8 2024. doi: 10.1109/WACV57701.2024.00187.

[70] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Proc. 31st Int. Conf. Neural Information Processing Systems (NIPS 2017)*, pages 5998–6008, Long Beach, CA, USA, Dec. 4–9 2017. doi: 10.5555/3295222.3295349.

[71] H.-H. Nguyen. Enhancing sentiment analysis on social media data with advanced deep learning techniques. *International Journal of Advanced Computer Science and Applications*, 15(5):98, 2024. ISSN 2156-5570. doi: 10.14569/IJACSA.2024.0150598.

[72] H. Wu, Y. Chen, W. Zhu, Z.-N. Cai, A. A. Heidari, and H. Chen. Feature selection in high-dimensional data: an enhanced rime optimization with information entropy pruning and dbscan clustering. *International Journal of Machine Learning and Cybernetics*, 15(9):4211–4254, Apr. 2024. doi: 10.1007/s13042-024-02143-1.

[73] P. Doucet, B. Estermann, T. Aczel, and R. Wattenhofer. Bridging diversity and uncertainty in active learning with self-supervised pre-training. volume arXiv preprint abs/2403.03728, Mar. 2024.

[74] H. Ding, H. Yu, and Z. Wang. Greedy strategy works for k-center clustering with outliers and coreset construction. *Proc. 27th Annual European Symposium on Algorithms (ESA 2019)*, pages 40:1–40:16, Sept. 9-11 2019. doi: 10.4230/LIPIcs.ESA.2019.40.

[75] Y. LeCun, C. Cortes, and C. J. C. Burges. The mnist dataset of handwritten digits (images) — pymvpa 2.6.5.dev1 documentation. Online resource, Apr. 9 2019. URL http://www.pymvpa.org/datadb/mnist.html. Accessed: Feb. 7, 2022.

[76] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical Report UTML TR 2009-001, University of Toronto, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf. Accessed: 2025-10-13.

[77] P. Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *ArXiv Preprint arXiv:1804.03209*, Apr. 2018.

[78] A. Chaudhry, M. A. Ranzato, M. Rohrbach, and M. Elhoseiny. Efficient lifelong learning with a-gem. in Proc. ICLR 2019 (Poster) New Orleans, LA, USA, May 2019. abs/1812.00420.

[79] F. Zenke, B. Poole, and S. Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pages 3987–3995, Sydney, Australia, Aug. 6–11 2017. PMLR.

[80] A. Kutalev and A. Lapina. Stabilizing elastic weight consolidation method in practical ml tasks and using weight importances for neural network pruning, Sep. 2021. 10.48550/arXiv.2109.10021.

[81] W. Baptiste, P. Denis, S. Olympieff, and S. Huet. Online class incremental learning with one-vs-all classifiers for resource constrained devices. page 1–6, 2023. doi: 10.1109/ISPA58351.2023.10279826.

[82] Savya K., Zhen Z., and Yifei H. Survey on memory-augmented neural networks: Cognitive insights to ai applications. *arXiv preprint arXiv:2312.06141*, 2023.

[83] D. Lopez-Paz and M. A. Ranzato. Gradient episodic memory for continual learning. In *Proc. 31st Int. Conf. Neural Information Processing Systems (NIPS 2017)*, pages 6467–6476, Long Beach, CA, USA, Dec. 4–9 2017.

[84] P. Sundaramoorthy, G. K. Gudur, M. R. Moorthy, R. N. Bhandari, and V. Vijayaraghavan. Harnet: Towards on-device incremental learning using deep ensembles on constrained devices. In *Proceedings of the 2nd International Workshop on Embedded and Mobile Deep Learning (EMDL '18), co-located with ACM MobiSys 2018*, pages 31–36, Munich, Germany, 2018. doi: 10.1145/3212725.3212728.

[85] J. Yoon, E. Yang, J. Lee, and S. J. Hwang. Lifelong learning with dynamically expandable networks. In *Proc. 6th International Conference on Learning Representations (ICLR '18)*, Vancouver, BC, Canada, Apr. 30–May 3 2018.

[86] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, Aug. 6–11 2017. doi: 10.5555/3305381.3305498.

[87] Y. Bao and W. Chen. Il4iot: Incremental learning for internet-of-things devices. In *Proc. 15th European Conference on Ambient Intelligence (AmI 2019)*, Lecture Notes in Computer Science, vol. 11912, pages 92–107, Rome, Italy, Nov. 13–15 2019. Springer. doi: 10.1007/978-3-030-34255-5-7.

[88] S. Gai, S. Lyu, H. Zhang, and D. Wang. Continual reinforcement learning for quadruped robot locomotion. *Entropy*, 26(1):93, 2024. doi: 10.3390/e26010093.

[89] A. Maekawa, H. Kamigaito, K. Funakoshi, and M. Okumura. Generative replay inspired by hippocampal memory indexing for continual language learning. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 930–942, Dubrovnik, Croatia, May 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.eacl-main.65.

[90] J. López, P. Sánchez-Vilariño, R. Sanz, and E. Paz. Implementing autonomous driving behaviors using a message driven petri net framework. *Sensors*, 20(2), 2020. ISSN 1424-8220. doi: 10.3390/s20020449.

[91] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017. ISSN 1091-6490. doi: 10.1073/pnas.1611835114.

[92] B. Sudharsan, P. Yadav, J. G. Breslin, and M. I. Ali. Train++: An incremental ml model training algorithm to create self-learning iot devices. In *Proceedings of the 18th IEEE International Conference on Ubiquitous Intelligence and Computing (UIC 2021)*, page 457–464, Newcastle upon Tyne, UK, Oct. 18–21 2021. doi: 10.1109/UIC54348.2021.00087.

[93] S. Younis and B. Seeger. Data-free generative replay for class-incremental learning on imbalanced data. volume abs/2406.09052, 2024. doi: 10.48550/arXiv.2406.09052.

[94] Z. A. Siddiqui and Md M. R. Islam. Progressive convolutional neural network for incremental learning. *IEEE Access*, 2021. doi: 10.1109/ACCESS.2021.3062465.

[95] M. Nikdan, T. Pegolotti, E. Iofinova, E. Kurtic, and D. Alistarh. Sparseprop: Efficient sparse backpropagation for faster training of neural networks at the edge. *International Conference on Machine Learning*, pages 26215–26227, 2023. ISSN 2640-3498.

[96] M. Ortiz, A. Cristal, E. Ayguadé, and M. Casas. Low-precision floating-point schemes for neural network training. *arXiv Preprint arXiv1804.05267*, 2018.

[97] Intel Labs. Taking neuromorphic computing to the next level with loihi 2 – technology brief. Technology brief, Sep. 2021. URL https://www.intel.com/content/www/us/en/research/neuromorphic-computing-loihi-2-technology-brief.html. Accessed: 2025-01-13.

[98] C. Silvano, D. Ielmini, F. Ferrandi, L. Fiorin, S. Curzel, L. Benini, F. Conti, A. Garofalo, C. Zambelli, E. Calore, S. F. Schifano, M. Palesi, G. Ascia, D. Patti, N. Petra, D. De Caro, L. Lavagno, T. Urso, V. Cardellini, G. C. Cardarilli, R. Birke, and S. Perri. A survey on deep learning hardware accelerators for heterogeneous hpc platforms. *ACM Computing Surveys*, 57(11):1–39, 2025. doi: 10.1145/3729215.

[99] S. Shi, Xiaowen C., K. C. Cheung, and S. See. Understanding top-k sparsification in distributed deep learning. *arXiv Preprint arXiv 1911.08772*, 2019.

[100] M. Kirtas, N. Passalis, A. Oikonomou, M. Moralis-Pegios, G. Giamougiannis, A. Tsakyridis, G. Mourgias-Alexandris, N. Pleros, and A. Tefas. Mixed-precision quantization-aware training for photonic neural networks. *Neural Computing and Applications*, 35(29):21361–21379, Aug. 2023. doi: 10.1007/s00521-023-08848-8.

[101] B. Vogginger, A. Rostami, V. Jain, S. Arfa, A. Hantsch, D. Kappel, M. Schäfer, U. Faltings, H. A. Gonzalez, C. Liu, C. Mayr, and W. Maaß. Neuromorphic hardware for sustainable ai data centers. *arXiv preprint abs/2402.02521*, Feb. 2024.

[102] G. Hinton, L. Deng, D. Yu, G. Dahl, A.–R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, Nov. 2012. doi: 10.1109/MSP.2012.2205597.

[103] NVIDIA. Train with mixed precision. https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html, Feb. 2023. Accessed 2023-02-01.

[104] M. Chahine, R. Hasani, P. Kao, A. Ray, R. Shubert, M. Lechner, A. Amini, and D. Rus. Robust flight navigation out of distribution with liquid neural networks. *Science Robotics*, 8(77):eadc8892, 2023. doi: 10.1126/scirobotics.adc8892.

[105] V. Jahmunah, E. Y. K. Ng, R.-S. Tan, S. L. Oh, and U. R. Acharya. Explainable detection of myocardial infarction using deep learning models with grad-cam technique on ecg signals. *Computers in Biology and Medicine*, 146:105550, 2022. doi: 10.1016/j.compbiomed.2022.105550.

[106] Y. LeCun, C. Cortés, and C. J. C. Burges. Mnist handwritten digit database. Online dataset, 2010. URL https://yann.lecun.com/exdb/mnist/. Accessed: 2025-01-04.

[107] J. Anitha Ruth, R. Uma, A. Meenakshi, and P. Ramkumar. Meta-heuristic based deep learning model for leaf diseases detection. *Neural Processing Letters*, 54(6): 5693–5709, June 2022. doi: 10.1007/s11063-022-10880-z.

[108] B. Zhao and H. Bilen. Dataset condensation with distribution matching. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 6514–6523, Waikoloa, HI, USA, Jan. 2023. doi: 10.1109/WACV.2023.00321.

[109] V. Lomonaco and D. Maltoni. Core50: a new dataset and benchmark for continuous object recognition. In *Proceedings of the 1st Annual Conference on Robot Learning (CoRL 2017)*, volume 78 of *Proceedings of Machine Learning Research*, pages 17–26. PMLR, Nov. 13–15 2017.

[110] K. J. Piczak. Esc: Dataset for environmental sound classification. In *Proceedings of the 23rd Annual ACM International Conference on Multimedia (ACM MM '15)*, pages 1015–1018, Brisbane, Australia, Oct. 13–17 2015. doi: 10.1145/2733373.2806390.

[111] D. Garcia-Gonzalez, D. Rivero, E. Fernandez-Blanco, and M. R. Luaces. A public domain dataset for real-life human activity recognition using smartphone sensors. *Sensors, Volume 20, 3, Basel, Switzerland*, 2020. doi: 10.3390/s20082200.

[112] X. Sun, X. Ren, S. Ma, and H. Wang. meprop: Sparsified back propagation for accelerated deep learning with reduced overfitting. In *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*, volume 70 of *Proceedings of Machine Learning Research*, pages 3299–3308. PMLR, Jun. 6-11 2017.

[113] Z. Zhang, P. Yang, X. Ren, Q. Su, and X. Sun. Memorized sparse backpropagation. *in Neurocomputing*, 415:397–407, 2020. doi: 10.1016/j.neucom.2020.08.055.

[114] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2018)*, pages 4510–4520, Piscataway, NJ, 2018. IEEE. ISBN 978-1-5386-6420-9. doi: 10.1109/CVPR.2018.00474.

[115] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, Sep. 1951. doi: 10.1214/aoms/1177729586.

[116] DCASE Community. Dcase 2020 challenge – detection and classification of acoustic scenes and events. http://dcase.community/challenge2020/index, 2022. Accessed: 2022-02-01.

[117] M.-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *Proc. 6th Indian Conference on Computer Vision, Graphics Image Processing (ICVGIP 2008)*, pages 722–729, Bhubaneswar, India, Dec. 16–19 2008. doi: 10.1109/ICVGIP.2008.47.

[118] L. Bossard, M. Guillaumin, and L. Van Gool. Food-101 – mining discriminative components with random forests. In *Proceedings of the 13th European Conference on Computer Vision (ECCV 2014)*, Lecture Notes in Computer Science, vol. 8694, pages 446–461. Springer International Publishing, Cham, Switzerland, 2014. doi: 10.1007/978-3-319-10599-4-29.

[119] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. doi: 10.1038/323533a0.

List of Figures

List of Tables

Appendix C

Abbreviation

Abbreviation	Definition
AI	Artificial Intelligence
AR	Augmented Reality
CIFAR	Canadian Institute for Advanced Research
CNN	Convolutional Neural Network
CORe50	Continual Object Recognition (50 classes)
DDA	Differentiable Data Augmentation
DEN	Dynamic Expandable Networks
DL	Deep Learning
DNN	Deep Neural Network
DPW	Discard Percentage Window
DQN	Deep Q-Network
DRIP	Data Retention Importance Protocol
Edge ML	Edge Machine Learning
ELL	Embedded Learning Library
EWC	Elastic Weight Consolidation
FedAvg	Federated Averaging
FLOPs	Floating Point Operations
GAN	Generative Adversarial Networks
GEM	Gradient Episodic Memory
Grad-CAM	Gradient-weighted Class Activation Mapping

Table C.1: Glossary of Abbreviations (Part 1)

Abbreviation	**Definition**
GPU	Graphics Processing Unit
HAR	Human Activity Recognition
HARNet	Human Activity Recognition Network
IPC	Images Per Class
IoT	Internet of Things
LLN	Low Power and Lossy Networks
MAML	Model-Agnostic Meta-Learning
MANNs	Memory-Augmented Neural Networks
MCU	Microcontroller Unit
ML	Machine Learning
MNIST	Modified National Institute of Standards and Technology
MNetV2	MobileNetV2 Architecture
NLP	Natural Language Processing
PD	Plant Disease (Dataset)
PPO	Proximal Policy Optimization
R&D	Research and Development
RAT	Radio Access Technology
SC	Speech Commands (Dataset)
SGD	Stochastic Gradient Descent
SI	Synaptic Intelligence
SoC	System on Chip
TFLM	TensorFlow Lite Micro
TinyProp	Sparse Backpropagation Algorithm Optimized for Edge ML
TinyPropv2	Enhanced Version of TinyProp
TPU	Tensor Processing Unit
WSN	Wireless Sensor Networks

Table C.2: Glossary of Abbreviations (Part 2)

Appendix D

Glossary

Term	Definition
Adaptive Top-k	A method in TinyProp where the number of gradients to update is dynamically adjusted based on local error.
Artificial Intelligence (AI)	A branch of computer science focused on creating machines capable of intelligent behavior.
Augmented Reality (AR)	An interactive experience that combines real-world and computer-generated content.
Backward Propagation	The process of updating the model's parameters by calculating gradients of the loss function.
Catastrophic Forgetting	The loss of previously learned information when a model is incrementally trained on new data.
Data Distillation	A process to compress large datasets into smaller subsets for more efficient model training.
Dataset Distillation	The process of reducing dataset size while preserving essential features for training, especially for on-device learning.
Deep Learning (DL)	A subfield of ML involving neural networks with many layers, used for more complex tasks like image recognition.
Edge Machine Learning (Edge ML)	The deployment of machine learning models on resource-constrained devices to process data at the source.
Elastic Weight Consolidation (EWC)	A regularization technique in incremental learning that selectively updates network weights and penalizes changes to those important for previous tasks.

Table D.1: Glossary of Terms - Part 1

Term	**Definition**
Federated Learning	A machine learning approach where models are trained across multiple decentralized devices while keeping data localized.
Forward Propagation	The process in a neural network where input data passes through layers to produce an output.
Generative Replay	A technique used in incremental learning where a generative model produces synthetic data to prevent catastrophic forgetting.
Gradient Episodic Memory (GEM)	A continual learning technique that uses a small memory buffer to store past data and replay it during training.
Grad-CAM	A visualization technique used to highlight regions of an input image that contribute most to a network's prediction.
Incremental Learning	A method where models continuously learn from new data without full retraining.
Internet of Things (IoT)	A network of devices connected via the internet, allowing them to collect and exchange data.
Low Power and Lossy Networks (LLN)	Networks characterized by high packet loss, low bandwidth, and instability, typically used in IoT systems.
Machine Learning (ML)	A subset of AI that allows systems to learn and improve from experience without being explicitly programmed.
Microcontroller Unit (MCU)	A small, low-cost embedded system designed for controlling other hardware.

Table D.2: Glossary of Terms - Part 2

Term	**Definition**
Mutual Information	A measure that quantifies the amount of information obtained about one variable through another.
On-device Training	Training of machine learning models directly on edge devices to enable real-time learning and decision-making.
Pruning	A technique used in neural networks to remove redundant or insignificant parameters, reducing model size without performance loss.
Quantization	A model compression technique that reduces the precision of weights and activations to improve computational efficiency.
Radio Access Technology (RAT)	A component of mobile telecommunication that connects devices to a cellular network.
Sparse Backpropagation	A method that selectively updates the most important gradients during backpropagation to reduce computational load.
Synaptic Intelligence (SI)	An incremental learning method that tracks the importance of weights over time to protect those important for previous tasks.
System on Chip (SoC)	An integrated circuit that integrates all components of a computer or other electronic systems.
TinyProp	A sparse backpropagation algorithm that dynamically adjusts the number of gradient updates per layer, optimized for Edge ML.
Tinyprop	An enhanced version of TinyProp, introducing a decision mechanism to skip unnecessary training steps.
Top-k Sparsification	A technique that retains only the top k% of gradients with the highest magnitude, reducing computational requirements.
Wireless Sensor Networks (WSN)	Networks formed using embedded devices that sense and transmit data wirelessly.

Table D.3: Glossary of Terms - Part 3