論文 / 著書情報
Article / Book Information

題目(和文)	高効率・高精度なエッジAIアンサンブル手法の研究
Title(English)	A Study on Efficient and Accurate Ensemble Methods for Edge AI
著者(和文)	熊澤峻悟
Author(English)	Shungo Kumazawa
出典(和文)	学位:博士(学術), 学位授与機関:東京科学大学, 報告番号:甲第382号, 授与年月日:2025年3月26日, 学位の種別:課程博士, 審査員:本村 真人,髙橋 篤司,CHU VAN THIEM,ISLAM A K M MAHFUZUL,佐々木 広
Citation(English)	Degree:Doctor (Academic), Conferring organization: Institute of Science Tokyo, Report number:甲第382号, Conferred date:2025/3/26, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Doctoral Dissertation

A Study on Efficient and Accurate Ensemble Methods for Edge AI

Department of Information and Communications Engineering
School of Engineering
Institute of Science Tokyo

February 2025

Author
Shungo Kumazawa

Supervisor
Prof. Masato Motomura

Doctoral Dissertation

A Study on Efficient and Accurate Ensemble Methods for Edge AI

Shungo Kumazawa

Abstract

Advancements in low-cost, high-performance device technologies have driven the rapid proliferation of IoT devices, expanding their applications across various domains such as healthcare, urban infrastructure, and industrial automation. This surge has brought attention to edge AI, which enhances responsiveness and energy efficiency by enabling distributed data processing at the network edge. As the number of IoT devices continues to grow in edge AI environments, efficiently utilizing these expanding device networks has become increasingly important. In particular, collaborative inference methods, which leverage multiple devices to overcome the limitations of individual device performance, are gaining significance.

One promising approach is Edge Ensembles, which achieve higher accuracy than a single device by integrating predictions across multiple devices based on an ensemble approach. They offer key advantages such as scalability to accommodate increasing IoT devices, efficient resource utilization through dynamic load balancing, and reduced reliance on centralized servers. Additionally, their adaptability to dynamic environments allows seamless handling of device mobility and availability changes, ensuring robustness in real-world scenarios. However, increasing the number of collaborating devices in cross-device ensemble inference can lead to higher communication overhead and latency. Additionally, fluctuations in device availability may cause variations in inference accuracy. To mitigate these challenges, this study addresses two key issues to improve accuracy and efficiency, enabling high-precision inference even with a reduced number of available devices. The first is the need for lightweight machine learning (ML) methods for both training and inference that can efficiently run in highly resource-constrained environments while achieving high accuracy through ensemble techniques. As IoT devices proliferate, reducing server dependency is essential not only for inference but also for model updates, ensuring the adaptability of Edge Ensembles. Second, using computationally intensive models like deep learning (DL) for complex tasks in Edge Ensembles significantly increases the inference cost compared to traditional ML models. This leads to prolonged inference

times per device, reducing the overall availability of devices within the ensemble cluster and potentially undermining the ensemble's benefits.

To overcome these limitations, this study introduces two novel proposals. The first is "Extra Ferns," a lightweight fern-based ensemble training method designed for resource-constrained edge environments. Extra Ferns significantly enhances inference accuracy compared to existing fern-based methods while consuming low energy consumption during training, making it particularly well-suited for edge device deployment. At its core, Extra Ferns represents a unique subset of decision tree ensembles, combining the strengths of existing methods while addressing their limitations. Specifically, it significantly decreases memory access and optimizes each tree in parallel, benefiting from the advantages of both Extra Trees and Random Ferns. As Extra Trees does, it generates nodes by randomly selecting attributes and generating thresholds. Then, as Random Ferns does, it builds ferns, which are decision trees that share identical nodes at each depth. In contrast to other ensemble methods using greedy optimization, Extra Ferns attempts global optimization of each fern. Experimental results show that Extra Ferns requires only 4.3% and 4.1% memory access for training models with 3.0% and 1.2% accuracy drops compared with Random Forest and Extra Trees, respectively. This study also proposes applying lightweight random projection to Extra Ferns as a preprocessing step, which improved further accuracy by up to 2.0% for image datasets.

The second proposal focuses on improving the efficiency and accuracy of model integration for DL-based Edge Ensembles. Enhancing model integration reduces the number of devices required to achieve the desired accuracy in collaborative inference tasks. This reduction in device usage leads to lower overall resource consumption across the cluster, enabling more efficient collaboration. To explore this improvement, this study investigates the effectiveness of conventional model integration techniques, such as cascade, weighted averaging, and test-time augmentation (TTA), when applied to Edge Ensembles to enhance their performance. Furthermore, this study proposes enhancements of these techniques tailored for Edge Ensembles. The cascade reduces the number of models required for inference but worsens latency by sequential inference processing. An increase in latency deteriorates real-time performance, posing significant challenges for its application in scenarios that demand immediate predictions. To address this issue, this study proposes m-parallel cascade, which reduces latency by adjusting the number of models processed simultaneously to m. This study also proposes learning TTA policies and weights for weighted averaging using ensemble prediction labels instead of ground truth labels. In the experiments, this study verified the effectiveness of each technique for the context of Edge Ensembles. The proposed m-parallel cascade achieved a 2.8 times

reduction in latency compared to the conventional cascade, even with a 1.06 times increase in computational costs. Additionally, the ensemble label-based learning demonstrated comparable effectiveness to the approach using ground truth labels.

This study advances collaborative inference methods through two key contributions. The first contribution enhances the practicality of fern-based Edge Ensembles in environments with extremely limited computational resources. Additionally, it enables these methods to perform training directly within edge environments, facilitating more adaptive and environment-specific inference. The second contribution strengthens the collaborative capabilities of DL-based Edge Ensembles, improving the overall performance of Edge Ensemble systems and demonstrating the feasibility of optimizing model integration in edge environments using only inference data. These contributions not only enhance the accuracy and efficiency of Edge Ensemble inference but also provide model training methods and analyses on collaborative frameworks, serving as valuable guidelines for designing practical cooperative inference systems.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, the rapid proliferation of low-cost, high-performance devices has fueled the growth of the Internet of Things (IoT), expanding its applications across diverse fields, including healthcare services, urban infrastructure, and industrial sectors [1, 2]. As a result, establishing an efficient framework for collecting and analyzing the vast amounts of data generated by numerous devices in real-time has become a critical challenge. Traditionally, IoT data analysis has often been performed centrally on the cloud. However, with the rise of edge computing, processing data in a distributed manner at the network edge offers potential solutions to challenges such as communication latency, bandwidth usage, and privacy protection [3]. Notably, "edge AI," which enables the execution of machine learning (ML) or deep learning (DL) models in edge environments, is gaining attention as a new paradigm that delivers intelligence on the edge side without relying on the cloud. This study focuses on improving the efficiency of edge AI inference methods that leverage multiple devices cooperatively, in preparation for an era with an even greater proliferation of IoT devices.

1.1 Motivation: The Need for Efficient Edge Inference in the Era of IoT

The number of IoT devices continues to grow, and further expansion is expected in the future [4, 5]. The forecast in [4], predicts that the compound annual growth rate (CAGR) of the IoT worldwide will reach 21% during the decade from 2020 to 2030, significantly increasing the number of connected devices. In this context, it is essential to develop edge inference systems that effectively utilize the increasing number of IoT devices.

Existing edge AI inference systems can be broadly categorized into three main approaches: (1) conducting inference directly on a single edge device, (2) utilizing an edge server to handle inference tasks, and (3) leveraging multiple devices or edge servers. (1)

approaches process the data acquired by the edge device directly on the device itself. This method eliminates the need for communication with other devices, resulting in low-latency responses. However, the limited computational resources of edge devices restrict their ability to run large models, making it challenging to achieve high accuracy in tasks requiring complex computations. In contrast, (2) approaches have advantages and disadvantages that differ from those of (1). It provides greater computational resources, enabling larger and more complex models. However, achieving rapid response times is more challenging due to the need for data transmission to the edge server. Furthermore, as the number of devices increases, the server's load grows significantly, leading to scalability issues caused by resource exhaustion and increased latency. Meanwhile, (3) approaches enable multiple devices or edge servers to collaborate for inference. One category of (3) involves collaboration between devices and servers for task offloading. These approaches combine the advantages of both (1) and (2), balancing computational efficiency and response times. They allow selective use of edge servers or edge devices depending on the complexity of the task, such as assigning simpler tasks to edge devices for low-latency responses and offloading more demanding computations to edge servers. However, like (2), this approach suffers from poor scalability as the number of devices increases, making it less effective in handling the anticipated growth of IoT devices. Another category in (3) focuses on collaboration among multiple edge devices without relying on a server. These methods coordinate multiple low-power edge devices to function as a larger computational resource. By adjusting the number of utilized devices, they can optimize the balance between processing speed and inference accuracy according to specific application requirements. As the number of IoT devices and the volume of data continue to grow, improving the inference performance of single devices remains important. However, to ensure scalability and efficiency, it is even more critical to develop methods that effectively coordinate multiple devices. Therefore, this study focuses on advancing methods within the (3) approach that leverage multiple devices to optimize performance and scalability.

Edge Ensembles [6] are collaborative inference methods that coordinate multiple devices to aggregate inference results based on the theory of ensemble methods. Combining the outputs from all participants improves overall accuracy compared to individual devices. These methods provide several advantages, including scalability, low dependency on external devices, optimized resource utilization across edge device groups, and resilience to device mobility. As such, these methods are promising approaches to address the anticipated growth in IoT devices, which is the focus of this study. However, the realization of Edge Ensemble presents challenges related to infrastructure, accuracy and efficiency, and security. Among these, enhancing the efficiency and accuracy of the ensemble methodology is a

crucial first step in demonstrating the feasibility of real-time and precise predictions within limited computational resources, thereby advancing the realization of Edge Ensemble inference. To achieve this, this study addresses the following two key issues.

1. The need for lightweight ML training/inference methods suitable for ensemble.

In Edge Ensembles, each device requires a fully executable inference model, necessitating lightweight ML models that can function in highly resource-constrained environments, as opposed to resource-intensive DL models. Additionally, as local training on individual devices enhances model diversity and increases the robustness of system-wide inference, lightweight edge-based training methods, alongside efficient inference, are highly desirable. Ensemble methods based on ferns, which are lightweight variants of decision trees, offer a potential solution to this challenge. However, existing fern-based methods lack sufficient accuracy. To overcome this limitation, this study proposes Extra Ferns, an edge-oriented fern-based ensemble training method. Extra Ferns achieves significantly higher accuracy than existing fern-based methods while reducing power consumption during the training compared to conventional decision tree-based methods. Its lightweight design in both training and inference makes it particularly well-suited for deployment on edge devices.

2. The need to elucidate more efficient ensemble methods in edge environments.

In Edge Ensembles, devices in idle or underutilized states are leveraged to perform inference simultaneously, and their prediction results are aggregated. Extending the idle periods of devices increases the number of devices available per unit of time within the cluster, thereby enhancing overall accuracy. To achieve longer idle periods, two strategies can be employed: reducing the inference time for each device or adopting a more efficient ensemble method that minimizes the number of devices required. DL is a powerful approach capable of achieving high accuracy in complex tasks, making it advantageous for use in Edge Ensembles. However, employing computationally intensive DL models in Edge Ensembles can lead to increased inference times for each device. Consequently, developing high-efficiency ensemble methods that minimize the number of devices required is crucial. To address this issue, this study explores efficient model integration methods for DL in Edge Ensembles by evaluating existing ensemble techniques and newly proposed methods in the context of Edge Ensemble.

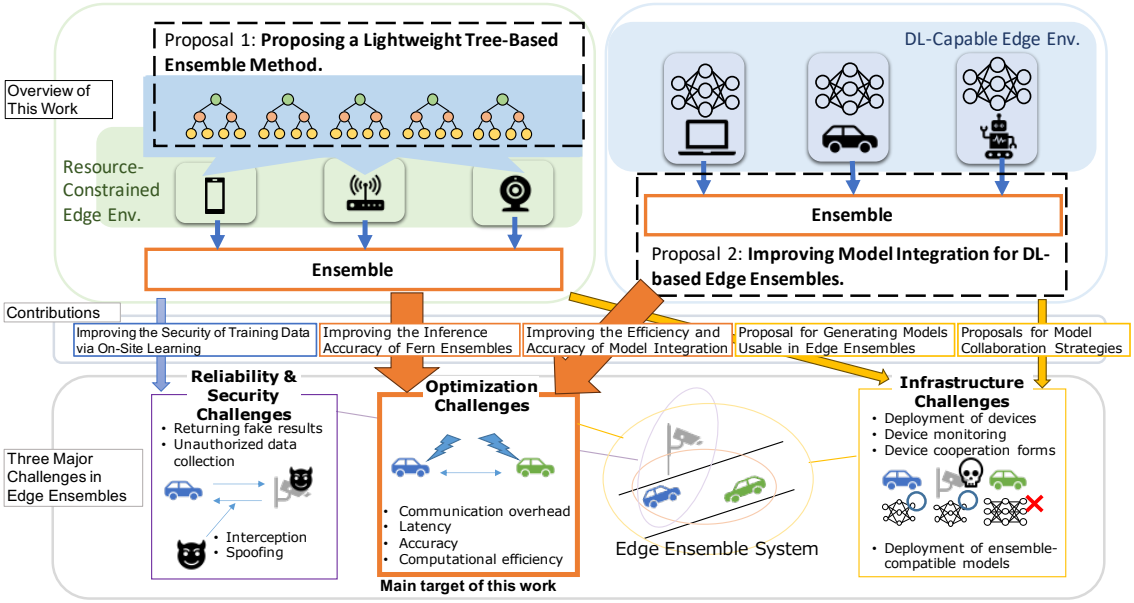

Fig. 1.1: Overview of this dissertation. The realization of Edge Ensembles faces three major challenges. This study primarily contributes to addressing issues related to accuracy and efficiency through two proposed approaches. Additionally, it partially contributes to reliability and security challenges, as well as infrastructure challenges.

1.2 Contributions and Outline of the Dissertation

This dissertation is divided into 5 chapters that present both practical and theoretical contributions to decision tree-based and deep learning-based ensemble methods and establish a new research approach at their intersection. Fig. 1.1 shows a summary of this study.

After this general introduction, Chapter 2 provides an overview of edge AI improvement techniques. It then explores methods for performing machine learning tasks collaboratively across multiple devices, with a detailed focus on the advantages of Edge Ensemble methods, which are the primary target of this study. The chapter further comprehensively explains ensemble methods and the core technology underlying Edge Ensembles. Finally, it discusses the current challenges in Edge Ensembles and outlines the direction of this research.

Chapter 3 proposes Extra Ferns, a new training method for fern-based ensemble models that are more lightweight versions of decision tree ensembles. The proposed method is designed with consideration for training in edge environments, reducing memory accesses, a major factor in power consumption, compared to decision tree-based training

methods. Extra Ferns significantly improves accuracy over traditional fern-based methods by optimizing the thresholds of each decision node in parallel. Furthermore, Extra Ferns sparsifies the leaf nodes of ferns, reducing memory usage by more than 90% compared to existing fern-based approaches. Additionally, Extra Ferns incorporates random projections to enhance the model's expressive power with minimal additional computational cost. This method demonstrated effective results on image datasets.

Chapter 4 introduces three model integration methods, cascade, Test Time Augmentation (TTA), and weighted averaging, to improve the efficiency of DL-based Edge Ensembles and evaluate their effectiveness. The chapter also proposes improved methods that are better suited to the context of Edge Ensembles. The first proposed method is m-parallel cascades. Traditional cascades process models one by one, but this method adjusts the number of models processed in parallel to m. By tuning m based on the required latency, it significantly reduces computational cost and latency compared to conventional Edge Ensembles. The second proposed method focuses on parameter tuning for TTA and weighted averaging using ensemble prediction labels. This approach demonstrates effectiveness comparable to conventional methods that use training data, enabling improved accuracy and efficiency in Edge Ensembles without relying on labeled data on the edge side.

Finally, Chapter 5 concludes the dissertation with a summary and a discussion.

Chapter 2

Background

This chapter begins with an overview of the development of edge AI, followed by a discussion on collaborative edge AI, where multiple devices or edge servers work together. Within collaborative edge AI, this study focuses on Edge Ensembles, highlighting their advantages and significance through comparisons with other approaches. Subsequently, the section delves into the theory and existing research on ensemble methods, which form the foundation of Edge Ensembles. Finally, it identifies the current challenges in Edge Ensembles and explains the research direction of this study.

2.1 Advancements in Edge AI

Machine learning on edge devices, so-called edge AI, is a natural progression given the advent of computationally powerful and efficient IoT devices. Unlike traditional cloud-based AI, edge AI addresses privacy and latency concerns by performing AI processing directly on edge devices. However, it necessitates computationally intensive machine learning tasks to be executed on devices with limited computing resources. Existing studies to address this challenge can be categorized into two main approaches.

The first approach focuses on improving the performance of individual devices. From a software perspective, techniques such as quantization [7, 8], pruning [9, 10], and knowledge distillation [11, 12] are employed to reduce the model size and computational complexity [13]. In DL, lightweight networks specifically designed for edge devices [14, 15] and neural architecture search [16–18] also provide effective solutions. From a hardware perspective, custom ASICs and FPGAs designed explicitly for AI tasks can significantly enhance computational efficiency [19]. These specialized accelerators are typically more energy-efficient than traditional CPUs and GPUs, which prioritize flexibility to support a wide range of workloads but incur higher energy consumption.

The second approach entails leveraging multiple devices to work collaboratively. This approach includes methods such as improving learning efficiency through the collaborative use of multiple devices [20, 21], partitioning networks across various devices or servers for inference [22, 23], and ensemble-based inference methods that integrate results from multiple devices [6, 24].

These two approaches are not mutually exclusive and can be effectively combined. For example, improving the efficiency of individual devices while leveraging their collaborative use can further enhance overall performance. However, the number of IoT devices is expected to increase significantly in the future. In such a scenario, the collaborative use of these growing IoT devices will become increasingly important. Therefore, this study focuses on the latter approach. The following section provides a detailed exploration of collaborative methodologies.

2.2 Collaborative Edge AI

This section begins with a comprehensive explanation of collaborative edge AI and concludes with an examination of the scope of this research. Collaborative edge AI can be categorized into two patterns: during training and during.

2.2.1 Collaborative Training

Collaborative methods during training primarily focus on protecting the training data stored on each edge device, rather than emphasizing real-time response. Federated learning [21] is a training method for machine learning models across multiple distributed devices while preserving the privacy of each device's local training data. Each device trains its local model using its own data and sends only the model updates, rather than the raw data, to a central server. The central server aggregates these updates to create a new global model, which is then distributed back to the devices. By iterating this process, a global model can train without transmitting the individual devices' data to external locations. Gossip learning [20] is similar to federated learning as a distributed learning method that preserves the privacy of training data by ensuring it remains within the node. However, unlike federated learning, gossip learning involves direct information exchange between nodes, eliminating the need for orchestration by a central server.

Privacy-preserving training plays an essential role in edge AI. However, since training requires significantly more computational resources than inference, it is generally conducted in environments such as edge servers rather than on small IoT devices in edge settings.

TABLE 2.1: Qualitative comparison between collaborate inference approaches (Reconstructed based on [24, Table 1]).

Collaborate	Method	Communications	Accuracy	Connectivity
Edge-cloud	Collaborate edge-cloud	Medium – sharing of compressed features	High – usage of large DNNs, possible distortion due to feature compression	Requires reliable link server/ specific devices
Multiple edge devices	Computing partitioning	High – due to repeated device-to-device communications		
	Edge Ensembles	Low – multi-casting of compressed features via device-to-device links	Adaptive – degraded in low connectivity, increases when collaboration is feasible	Fully adaptive – operable in different connectivity levels

2.2.2 Collaborative Inference

Collaborative methods during inference aim to enhance efficiency and enable real-time response through cooperation among edge devices or between edge devices and the cloud. The efficiency of edge inference is crucial because it directly addresses issues closely tied to user experience, such as real-time performance and energy efficiency. Moreover, since inference is performed more frequently than training, it has a significant impact from a practical perspective.

Collaborative inference involves sharing inference data or intermediate features with other devices or servers, compromising data privacy compared to inference conducted on a single device. However, the collaborative use of devices allows for the creation of large computational resources in edge environments. Collaborative inference can generally be divided into three main approaches [25]. Fig. 2.1 shows an overview of each method, and Table 2.1 summarizes a qualitative comparison between each approach.

Collaborate Edge-cloud inference

The most commonly used approach in collaborative inference between devices and servers involves partitioning the model between edge devices and servers to distribute the computational load. Edgent [26] partitions a neural network at the layer level between edge devices and edge servers to enable co-inference. Furthermore, on the edge side, the network incorporates multiple exit points, further reducing latency.

This approach enables the execution of large DNN models and achieves high accuracy by

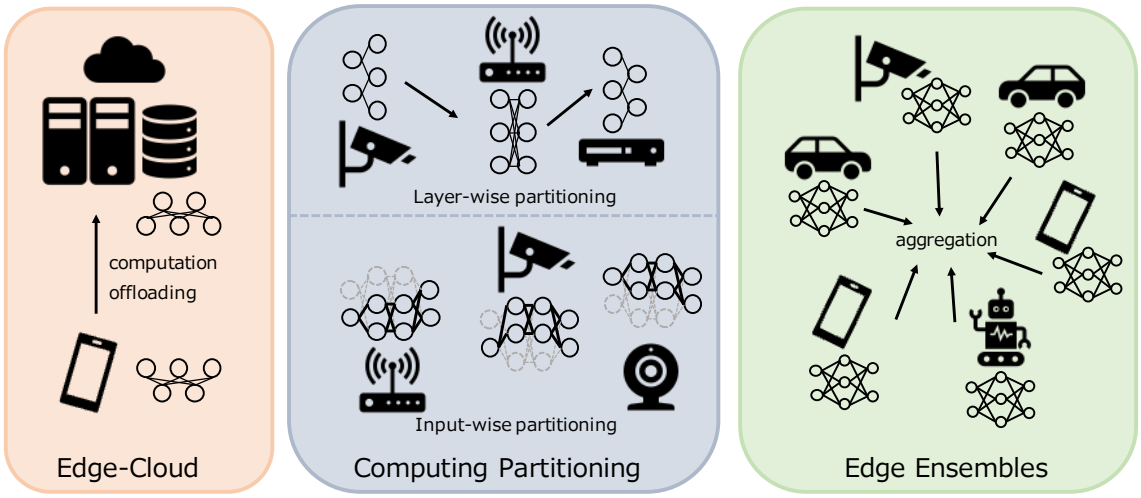

Fig. 2.1: Collaborative inference methods.

utilizing the server's computational resources. However, it requires a continuous connection with the server that hosts parts of the model. Therefore, this approach is particularly well-suited for handling computationally intensive data in fixed surveillance and monitoring devices, which are often static and feature highly reliable connectivity [24].

Computing Partitioning Among Edge Devices

The most straightforward method for computing partitioning involves dividing a single DNN model by layers and distributing its components across multiple edge devices [25, 27, 28]. AAIoT [22] optimizes inference by partitioning a neural network at the layer level across multiple edge devices rather than dividing the model between a server and devices, as in Edgent. This approach uses dynamic programming to allocate layers to each device efficiently. In addition to methods that partition the model by layers, some approaches divide input data along its dimensions. One example is MoDNN [23], which reduces the computational load on each device by partitioning the input data and performing parallel processing across multiple networks on each device.

While model partitioning allows establishing large networks on the edge by distributing parts of the network across multiple devices, it has certain drawbacks, such as the strong dependency on the availability of neighboring devices with required DNN partitions and the need for complex control of repeated device-to-device communications [24]. Computation Partitioning is well-suited for environments where devices operate within the same location, such as industrial IoT sensors in factories. These environments readily

facilitate orchestration and optimal DNN partitioning [24].

Edge Ensembles

Edge Ensembles utilize ensemble methods to aggregate the inference results from respective models deployed on each edge device, achieving higher accuracy than single-device inference. During the inference process, the local device shares the collected data with nearby devices. Each device conducts inference using the shared data and returns its predictions to the local device. Finally, the local device aggregates all predictions, including its own [6].

Unlike computing partitioning, Edge Ensemble cannot execute full networks entirely on the edge. However, it demonstrates high adaptability by increasing the number of devices participating in inference based on the required accuracy. Furthermore, since each device operates with a completely independent model, it exhibits low dependency on other devices and achieves high availability. Therefore, this approach best suits highly mobile environments, such as vehicular systems, where devices operate in diverse conditions, and collaboration is established ad hoc [24].

2.2.3 Summary

As demonstrated in the previous sections, each approach of collaborative edge AI excels in specific scenarios and plays a significant role in its respective context. However, as the number of IoT devices continues to grow, developing more scalable and efficient edge AI systems will become increasingly critical. Edge Ensembles are particularly well-suited to addressing the challenges associated with this IoT growth. Their decentralized and low-dependency architecture facilitates ad hoc collaboration among devices, enabling practical functionality in collaborative scenarios. This makes them especially suitable for environments such as smart cities, where IoT devices are densely distributed. Accordingly, this study focuses on exploring and advancing Edge Ensembles as a key method for collaborative inference. The following section provides a detailed explanation of ensemble methods, which form the fundamental theory behind Edge Ensemble.

2.3 Ensemble Methods

Edge Ensemble is a collaborative inference method that implements ensemble techniques as its theoretical foundation. This chapter begins by explaining the theory behind the

success of ensemble methods, followed by an overview of the primary types of ensemble approaches.

2.3.1 Theory of Ensembles

There are various interpretations of the success of ensemble methods. From a mathematical perspective, some studies explain the theory using a bias-variance-covariance decomposition to analyze ensemble errors and investigate how each component of the errors changes under different ensemble techniques [29]. For regression tasks, the expected squared error of the ensemble is expressed as follows:

$$E\left[(\bar{o}-t)^2\right] = \overline{\text{bias}}^2 + \frac{1}{M}\overline{\text{var}} + \left(1-\frac{1}{M}\right)\overline{\text{covar}}, \tag{2.3.1}$$

where t is the target value, $\bar{o}$ is the ensemble prediction, and M is the number of base learners in the ensemble. $\overline{\text{bias}}$, $\overline{\text{var}}$, and $\overline{\text{covar}}$ represent, respectively, the average prediction error of the base learners, the variance in predictions among the base learners, and the covariance of predictions between pairs of base learners as follows:

$$\begin{aligned}
\overline{\text{bias}} &= \frac{1}{M}\sum_i \left(E\left[o_i\right]-t\right), \\
\overline{\text{var}} &= \frac{1}{M}\sum_i E[o_i - E[o_i]]^2, \\
\overline{\text{covar}} &= \frac{1}{M(M-1)}\sum_i \sum_{j\neq i} E\left[\left(o_i - E[o_i]\right)\left(o_j - E[o_j]\right)\right],
\end{aligned} \tag{2.3.2}$$

where o_i indicates the output of i-th base learner. Covariance represents the strength of the relationship between base learners. Greater diversity among learners leads to negative covariance, suggesting that increasing model diversity, without altering bias or variance, can reduce the overall error. Furthermore, the number of base learners also impacts the result; as the number of learners increases, the influence of variance diminishes. This decomposition demonstrates that designing low-correlated individual learners can improve performance [30].

Additionally, Dietterich intuitively explained the effectiveness of ensemble learning from three perspectives: statistical, computational, and representational [31]. In this theory, model training algorithms are viewed as processes that identify the optimal hypothesis from a set of hypotheses in the space. The statistical reason is that when the hypothesis space is vast relative to the amount of training data, the training algorithm may select different

hypotheses that exhibit similar performance on the training data. Combining multiple hypotheses through ensemble methods reduce the risk of misclassification, enabling more stable predictions. The computational reason is that when a single learner gets trapped in a local optimum, exploring with different initial conditions can lead to discovering additional approximate solutions. In this case, integrating predictions can be seen as performing multiple searches, resulting in a better approximation of the true unknown function. The representational reason is that when none of the training methods can accurately represent the true function, combining multiple hypotheses can express complex functions that a single hypothesis cannot represent. This extends the representational space of the hypothesis set, enabling more accurate predictions.

2.3.2 Ensemble Learning Methods

Ensemble learning methods can primarily be classified into three types based on the model construction method: bagging, boosting, and stacking [29, 32].

Bootstrap aggregating (bagging) [33] is an ensemble method in which each base model is trained independently on random subsets of the training data to construct the overall ensemble model. Bagging enhances model diversity by overfitting each model on a random subset of the training data and achieves higher accuracy than a single model by aggregating their predictions. It is particularly effective for unstable models, such as decision trees, which are highly sensitive to variations in the training data.

The most commonly used method related to bagging is Random Forest. Random Forest is a decision tree-based ensemble method that, like standard bagging, trains each decision tree on random subsets of the dataset. However, it introduce additional randomness compared to standard bagging. When constructing decision trees in a Random Forest, each node determines its splitting condition by randomly selecting a subset of candidate attributes and optimizing the threshold to partition the data across those attributes effectively. The advantages of this method include not only enhancing model diversity and improving accuracy compared to standard bagging but also reducing computational costs by limiting the number of attributes considered when searching for optimal thresholds.

Boosting is a method that sequentially trains multiple weak models to enhance the overall performance of the ensemble. While bagging employs deep decision trees that tend to overfit random subsets of data, boosting utilizes shallow decision trees with reduced expressive power, designed to function as weak learners. During training, weights are assigned to both weak learners and data, dynamically updating them at each step to reflect

their relative importance. Model weights are calculated based on prediction accuracy, with higher weights assigned to weak learners with lower error rates, thereby increasing their influence on the final prediction. Simultaneously, data weights are increased for misclassified samples at each iteration, guiding new weak learners to focus on harder-to-classify data. This complementary interaction among the trees enables even shallow trees to achieve high performance, facilitating efficient model construction.

The main methods of boosting include Adaboost [34], Gradient Boosting Decision Tree (GBDT) [35], XGBoost [36], and CatBoost [37]. Adaboost strengthens weak learners sequentially by increasing the weights of misclassified data. GBDT improves accuracy by training weak learners to minimize a loss function using gradient descent. XGBoost builds on gradient boosting by enhancing its speed and adding regularization to reduce overfitting, making it widely used for large datasets and competitive programming. CatBoost is designed to handle categorical features efficiently without extensive preprocessing. It employs ordered boosting to mitigate overfitting and uses symmetric trees, known as ferns (or oblivious decision trees, decision tables), as base learners to achieve faster training and improved accuracy.

Stacked Generalization (stacking) [38] combines predictions from two or more base models by using a secondary ML model, referred to as a meta-learner. The base learners train using distinct machine learning algorithms or subsets of the training data attributes, and the meta-learner trains on their predictions as new features. The meta-learner integrates these predictions to produce the final output, leveraging the strengths of each base model while compensating for their individual weaknesses, thereby achieving higher overall accuracy.

2.3.3 Ensemble Integration Strategies

Ensemble methods encompass not only the base model training strategies outlined in the previous section but also a variety of strategies for model integration.

Averaging is one of the most commonly used model integration methods in ensemble learning. This method is typically used in cases like bagging, where base models have no significant performance differences. It computes the average of the outputs from all models for regression tasks to produce the final prediction. For classification tasks, it averages the predicted probabilities for each class across all models and selects the class with the

highest average probability as follows:

$$\text{class} = \arg\max_{c \in C} \sum_i P_i(c), \tag{2.3.3}$$

where $P_i(c)$ represents the predicted probability of the i-th model for the class c. Technically, the summed probabilities in the equation should be divided by the number of models to calculate the average. Still, since this does not change the result, it is omitted here. Another commonly used method is majority voting, in which each weak learner predicts a single class label, and the final output is determined as the most frequent class among the base models. This approach can be considered a special case of averaging, as it is equivalent to each model assigning a 100% probability to a single class during the averaging process.

Weighted Averaging is effective in heterogeneous ensembles, where base models vary in size or consist of diverse types of models. Unlike standard averaging, weighted averaging assigns weights to each model during the averaging process as follows:

$$\text{class} = \arg\max_{c \in C} \sum_i w_i P_i(c), \tag{2.3.4}$$

where w_i means the importance of the i-th model. These weights can be generated during training through boosting methods, set based on Bayesian theory [39–41], determined by the performance of each base learner [42–45], or assigned by a meta-learner through stacking methods [46].

Cascade is a method that performs inference sequentially with base models and terminates the process early if it determines that the prediction is sufficiently reliable. This approach reduces computational costs compared to using all models for inference. The original cascade [47] was designed to accelerate face recognition by enabling the rapid determination of whether a region is a background or a face. It significantly reduced computational costs by performing inference sequentially with models and terminating the process as soon as any model identified the region as a background. The soft cascade [48] is an improved version of the original cascade, which accumulates prediction information from each model during sequential inference, allowing decisions to be made based on the combined predictions of all preceding models. Unlike the original cascade, which discarded intermediate prediction information, the soft cascade retains and integrates this data, making it more comparable to ensemble methods.

Ensemble Selection is a method that selects a subset of models from an ensemble for inference. The base models in an ensemble are typically constructed using different

algorithms or distinct subsets of the training data, resulting in varying performance levels across models. While some models demonstrate strong predictive capabilities, others may perform poorly. Instead of combining all models indiscriminately, ensemble selection seeks to enhance the overall performance of the ensemble by selecting a subset of models with superior performance.

One of the most straightforward ensemble selection methods is Forward Stepwise Selection [49]. This method sequentially and greedily selects model combinations to maximize the ensemble's accuracy on the validation set. Other methods include selecting ensemble members based on the scores of each model calculated during training, identifying optimal members using optimization techniques, and dynamically selecting ensemble members based on the features of the input data.

2.3.4 Ensemble Methods for Deep Learning

Ensemble methods have a long-standing history in machine learning, with decision tree-based approaches traditionally dominating their applications. In recent years, however, their adoption within the field of DL has increased substantially. Researchers have explored various adaptations of classical ensemble techniques for DL, including strategies such as averaging multiple deep neural network (DNN) models, adapting boosting algorithms for DL frameworks, and employing DNN models as base learners in stacking architectures. Furthermore, novel ensemble techniques designed explicitly for DL and implicit ensemble methods arising from the unique structures of deep learning models have also been proposed. This section explains these new ensemble methods specific to DL.

Implicit Ensembles refer to a technique where a single model is trained to function as if it were an ensemble of multiple models. This approach eliminates the need to train multiple models individually, significantly reducing computational costs and memory usage while retaining the benefits of ensemble learning, such as improved generalization performance and mitigation of overfitting.

The most typical example of an implicit ensemble is Dropout [50], which prevents overfitting by randomly dropping out nodes during training. This means that each training iteration operates on multiple diverse submodels within a single model. During inference, using all the parameters allows the model to be interpreted as an ensemble of these submodels. Another commonly used example is ResNet [51], which introduces skip connections into the network structure to improve accuracy. These skip connections create multiple paths within a single network, which can be considered as subnetworks. The model can be interpreted as an ensemble of these subnetworks during inference.

Snapshot Ensemble [52] saves intermediate model parameters as 'snapshots' during the training process of a single neural network and utilizes them as an ensemble during inference. This approach enables the generation of multiple models within a single training session, eliminating the need to train multiple independent models.

It introduces a learning rate scheduling technique called Cyclic Learning Rate, which guides the model to converge to different local optima during training. This process ensures model diversity within a single training run and enhances the effectiveness of the ensemble.

Mixture of Experts (MoE) [53] is a network architecture that dynamically selects 'expert models' based on the input data. An MoE consists of multiple expert models, each specialized for specific subtasks or characteristics of the input data, and a gating network that analyzes the input to determine which experts to utilize dynamically. When receiving input data, the gating network computes a score for each expert, representing the degree of confidence assigned to them based on the input. Each expert processes the input and generates an output, which is then combined using a weighted averaging by the scores from the gating network to produce the final output. MoE is similar to ensemble selection but differs in that MoE trains the gating network and experts jointly. In contrast, ensemble selection typically determines the combination of base models after training them individually.

A commonly used variation is sparse MoE [54], which is effectively applied to large-scale models such as image recognition models [55] or large language models (LLMs) [56]. Sparse MoE reduces computational costs by making the score function produced by the gating network sparse, thereby limiting the number of selected experts. Rather than reducing the total number of experts, sparse MoE decreases the number of experts activated by the gating network, allowing the total parameter count to remain unchanged while significantly lowering computational requirements.

Test Time Augmentation (TTA) is a technique that applies data augmentation to test data to improve a model's predictive accuracy. Data augmentation, which involves transformations like rotation, flipping, and scaling, is typically employed during training in DL to increase data diversity, prevent overfitting, and improve a model's generalization and robustness. [57]. TTA extends this concept to the testing phase to further enhance the model's accuracy and robustness.

In the TTA process, test data undergoes various transformations to generate multiple augmented versions. The model performs inference on each transformed version independently, and the predictions are then averaged to produce a final result. This method modifies the data rather than the model itself, making it an ensemble technique that uses

multiple views of the same data without requiring multiple models.

2.4 Current Challenges for Edge Ensembles

As mentioned in Chapter 1.1, existing Edge Ensembles face two challenges. This section provides a detailed discussion of these challenges, explores potential solutions, and clarifies the research direction for subsequent chapters, incorporating related research introduced thus far.

1. The need for lightweight ML training/inference methods suitable for ensemble.

In Edge Ensembles, unlike computing partitioning, where each device holds a portion of the partitioned model, each device must have a fully functional model capable of performing inference independently. This approach enhances the availability of Edge Ensembles by reducing dependency on other devices. However, operating on devices with extremely limited computational resources requires ML models that are not only lightweight enough to function independently on such devices but also suitable for ensemble to combine their results effectively in edge. Furthermore, considering the expected increase in edge devices, it would be even more advantageous to have algorithms that allow these lightweight models to perform additional training on edge devices with slightly more available computational resources.

Fern, a lightweight variation of decision trees that shares nodes at the same depth, offers significant advantages in edge AI. These include reduced memory usage due to its minimal number of internal nodes, parallel processing of each node enabling low-latency inference, and compatibility with ensemble methods, which aligns well with decision tree-based models. Additionally, ferns have fewer internal nodes, significantly reducing the number of feature accesses required during their construction compared to decision trees. This reduction minimizes memory accesses during training, a primary power consumption factor in decision tree training. These characteristics make fern an ideal ML model for Edge Ensembles.

Several studies [37, 58, 59] have applied boosting to fern models. One of them, CatBoost [37], stands out for achieving particularly high accuracy. A key feature of CatBoost is ordered boosting, which processes training data in random order and ensures that each data point does not reference subsequent data points during training. This approach prevents target leakage, which occurs when correct answer labels in the training data influence the model, and mitigates prediction shift, which refers to the distribution mismatch between training and test data, enabling the construction of models with enhanced generalization

performance. As a result, CatBoost achieves higher accuracy than existing decision tree ensembles. However, since boosting algorithms optimize the predictive accuracy of the entire ensemble, challenges arise in Edge Ensembles if each device holds only a subset of the boosting models rather than the complete model set. This scenario can lead to issues similar to those seen in computing partitioning, making boosting algorithms less practical for Edge Ensembles. Furthermore, ordered boosting requires the management of multiple model versions for multiple random permutations, resulting in a computationally heavy training algorithm unsuitable for on-device training in Edge environments.

Random Ferns [60] is a fern-based bagging method specifically designed for image data. During training, each node selects two random pixels from the image and compares their values to determine the data splitting. Unlike boosting methods, bagging methods are well-suited for Edge Ensembles because they function effectively by simply aggregating the predictions from the models held by participating devices. Additionally, Random Ferns is a lightweight method that selects branching conditions randomly, making it suitable for on-device training in edge environments. However, it is limited to image data and cannot be applied to general-purpose datasets. To address this limitation, An extended method called rFerns adapts Random Ferns for general-purpose datasets. Despite this adaptation, rFerns is a highly random algorithm with few internal nodes, resulting in significantly lower accuracy than existing decision trees, even when ensembled. Additionally, there is a challenge of increased memory usage in the leaf nodes of ferns.

This study focuses on fern-based bagging methods to capitalize on their suitability for Edge Ensembles. Specifically, it proposes a new training method to address the accuracy and memory usage issues associated with existing approaches such as Random Ferns and rFerns. The goal is to establish lightweight ML training and inference methods suitable for ensembles on the edge.

2. The need to elucidate more efficient ensemble methods in edge environments.

DL is a powerful approach capable of achieving high accuracy, making it suitable for complex tasks in Edge Ensembles. However, the computational cost of DL inference is significantly higher than that of traditional ML inference, resulting in prolonged inference times per device. This issue can lower the overall availability of devices within an Edge Ensemble cluster, potentially diminishing the benefits of the ensemble. There are two primary strategies to address this challenge:

(I) **Reducing inference time on each device**: Shorter inference times directly improve the availability of devices within the Edge Ensemble cluster.

(II) **Establishing more efficient model integration methods for ensemble inference**: Efficient integration techniques can reduce the number of devices required for the ensemble, thereby enhancing the availability of devices within the cluster. Additionally, model integration methods that not only improve efficiency but also achieve higher accuracy can reduce the number of devices needed to attain the same level of precision.

Regarding the first strategy, as discussed in Section 2.1, numerous approaches have been proposed to achieve shorter inference times. However, the second strategy has not been thoroughly investigated, particularly in terms of comprehensive research into model integration methods tailored for Edge Ensembles. Establishing such methods is crucial for the development of Edge Ensembles.Therefore, this study addresses the second challenge, aiming to develop efficient model integration techniques for Edge Ensemble inference.

A model integration approach for enhancing the efficiency of Edge Ensembles can consider methods such as weighted averaging, cascade, and ensemble selection, as discussed in Section 2.3.3. Among these methods, using ensemble selection may result in devices hosting relatively more important models being preferentially included in the ensemble members. This could lead to an uneven distribution of computational load, with specific devices bearing a disproportionate burden, making ensemble selection unsuitable for Edge Ensemble scenarios.

In contrast, weighted averaging adopts a similar approach to ensemble selection but assigns weights to ensemble members instead of selecting specific members. This approach integrates predictions from models on all available devices according to their weights rather than selecting specific devices with higer weights and integrating predictions from them. This makes it suitable for Edge Ensembles scenario and offers the potential for accuracy improvement. However, Edge Ensembles are unique in their ability to operate even in dynamic environments where devices frequently join or leave. This creates a challenge in assigning weights to devices joining the ensemble without predefined weights. To address this issue, this study proposes a new method to assign weights to models without using training data on the edge devices. The details of this approach are discussed in Section 4.3.2.

Cascades reduce the number of models required for predictions in ensemble methods. In the context of Edge Ensembles, this approach is expected to lower the number of devices necessary for inference. However, while general Edge Ensembles allow devices to perform inference in parallel, cascades introduces sequential processing, which raises concerns about increased latency. To address this challenge, this study proposes a novel method to reduce the latency of cascades. The details of the proposed approach are discussed in

Section 4.3.1.

Additionally, TTA is another method expected to enhance accuracy when adapted for Edge Ensembles. Unlike methods aimed at improving model integration part in ensemble techniques, TTA is an ensemble approach applied to a single model. However, applying TTA to generate ensemble predictions from a single model and subsequently integrating these predictions from multiple models, as in standard ensemble methods, has demonstrated additional accuracy improvements [61]. Therefore, in the context of Edge Ensembles, performing TTA on each device and combining the results across devices could similarly enhance accuracy. However, since TTA increases inference time on individual devices, its overall effectiveness in Edge Ensemble scenarios remains uncertain. This study evaluates the utility of TTA in Edge Ensembles through experiments detailed in Section 4.4.3.

Chapter 3

Extra Ferns: An Edge-oriented Fern Ensemble Learning Method with Minimal Memory Access

This chapter introduces Extra Ferns, a lightweight ensemble learning method tailored for edge environments, where both training and inference are resource-constrained. Extra Ferns employs ferns as base learners, which are even smaller than traditional decision trees, making it particularly suitable for ensemble inference in edge environments with extremely limited computational resources. Additionally, Extra Ferns offers a more energy-efficient training method compared to decision tree ensembles, enabling more feasible training in resource-constrained edge environments.

3.1 Introduction

Machine learning on edge devices, so-called edge AI, is a natural progression given the advent of computationally powerful and efficient IoT devices. The most typical form of edge AI is load balancing: training on clouds and inference on the edge. By reducing data transmission from an edge to clouds, edge AI provides a solution for privacy, high power consumption, and low bandwidth issues. However, edge AI often requires on-site training with the data obtained from edge devices because training on clouds requires massive data transmission from the edge to clouds, resulting in increased power consumption. Then, why is on-site training less common? What makes it challenging?

The challenge is that the computational cost for training is unaffordable to edge devices; training on the edge is not advantageous compared with training on clouds, even considering the data transmission. For example, neural networks require significant computational

capabilities and large memory for their training based on back-propagation. However, it is difficult for edge devices to satisfy these requirements or to afford the power consumption required for such computation. Therefore, machine learning algorithms with more lightweight but efficient training methods are required for edge devices.

To tackle this problem, this work proposes an edge-oriented machine learning algorithm called *Extra Ferns* which is based on conventional decision tree ensemble learning methods [60, 62–64] that use decision trees as base learners: Random Forest [62], extremely randomized trees (Extra Trees) [63], Random Ferns [60], and rFerns [64]. Decision tree ensembles are not state-of-the-art machine learning algorithms, but they still outperform neural networks for tabular data. Compared with neural networks, they require only a few parameters and a small amount of computation. These are suitable features for training on the edge, but there is still room for improvement in the training phase in terms of power consumption and processing performance.

Extra Ferns is an algorithm that takes advantage of the benefits of both Extra Trees and Random Ferns. As Extra Trees does, it generates nodes by randomly selecting attributes and generating thresholds. This node generation largely reduces off-chip memory accesses, which are the dominant factor in power consumption. Then, as Random Ferns does, it builds ferns, which are decision trees sharing identical nodes at each depth. This fern structure eliminates the processing dependency on input data. Also, by using this structure, Extra Ferns conducts non-greedy optimization in training to find optimal ferns.

The contributions of this chapter can be summarized as follows:

- This work describes the details of Extra Ferns, which requires an extremely small number of off-chip memory accesses.
- Extra Ferns conducts non-greedy optimization in parallel, which leads to accuracy improvement.
- This work also empirically shows that Dirichlet prior, as used in Random Ferns, is dispensable for Extra Ferns.
- Finally, this work proposes applying random projection [65, 66] to Extra Ferns as a preprocessing step for further accuracy improvement.

The remainder of this paper is organized as follows. Section 3.2 briefly describes the features of conventional decision tree ensembles. Section 3.3 describes the details of the proposed Extra Ferns based on them, and then Sections 3.4 presents preliminary experimental results and evaluation results, respectively. Section 3.5 describes how to

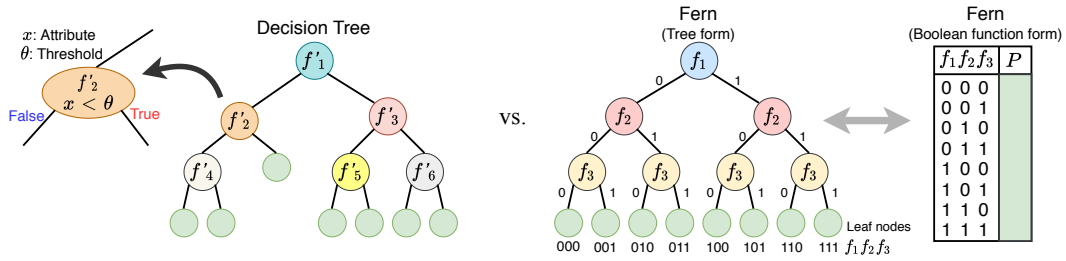

Fig. 3.1: Structure of a decision tree and fern. A decision tree consists of internal nodes, branches, and leaf nodes, which represent tests on attributes, the tests' outcomes, and expected values, respectively. A fern is a unique subset of decision trees where all nodes at the same depth are identical. As a result, a fern can be represented as a Boolean function.

introduce random projection to Extra Ferns for accuracy improvement and demonstrates its effectiveness through experimental results. Finally, Section 3.6 concludes this paper.

3.2 Decision Tree Emsembles

Decision tree ensembles are statistical machine learning algorithms that use decision trees as base learners. As shown in Fig. 3.1, a decision tree consists of internal nodes, branches, and leaf nodes, which represent tests on attributes, the tests' outcomes, and class labels/expected values, respectively. Ferns are decision trees with a unique structure, sharing identical internal nodes at each depth. Therefore, ferns can be considered Boolean functions using the internal nodes' outcomes as arguments.

This section briefly describes four ensemble algorithms used in designing Extra Ferns: Random Forest, Extra Trees, Random Ferns, and rFerns. Random Forest and Extra Trees use trees as base learners, while Random Ferns and rFerns use ferns. Table 3.1 provides a qualitative comparison of ensemble algorithms, and Table 3.2 presents a comparison of their training complexities. Table 3.3 defines the variables used in Table 3.2. The following sections explain the details of each algorithm separately.

Random Forest

Random Forest builds a forest, i.e., a set of decision trees, based on randomized node optimization [62]. It makes each tree composing the forest with $\tilde{N}$ bootstrapped samples from N training data. Once Random Forest selects a training dataset to build a tree, it randomly samples K attributes and chooses the best internal node to maximize the

TABLE 3.1: Qualitative Comparison Between Ensemble Algorithms.

Algorithm	Base Learner	Bootstrap Sampling	Split Point Selection	Supported Data
Random Forest	Tree	Yes	Partially Optimized – search the optimal threshold among random K attributes	ALl Types
Extra Trees	Tree	No	Partially Random – choose the best one from several random nodes	ALl Types
Random Ferns	Fern	No	Fully Random	Image Only
rFerns	Fern	Yes	Fully Random	All Types
Extra Ferns	Fern	No	Partially Optimized – use parallel threshold optimization for random nodes	All Types

TABLE 3.2: Complexity Comparison between Ensemble Algorithms.

Algorithm	# Leaf Nodes	# Internal Nodes	# Training Data Accesses	Computational Complexity
Random Forest	$O(M\tilde{N})$	$O(M\tilde{N})$	$O(KM\tilde{N}\log\tilde{N})$	$O(KM\tilde{N}(\log\tilde{N})^2)$
Extra Trees	$O(MN)$	$O(MN)$	$O(KMN\log N)$	$O(KMN\log N)$
Random Ferns	$O(2^D M)$	$O(DM)$	$O(DMN)$	$O(DMN)$
rFerns			$O(DM\tilde{N})$	$O(DM\tilde{N})$
Extra Ferns			$O(UDMN)$	$O(UDMN\log N)$

TABLE 3.3: Definitions of Variables for Complexity Comparison.

Variable	Description
D	depth of trees / ferns
K	# of candidate attributes
M	# of trees / ferns
N	# of training data
$\tilde{N}$	# of bootstrapped training data
U	# of updates in Extra Ferns

infomation gain as following criterion:

$$G(S_p, f) = I(S_p) - \left\{ \frac{N_l}{N_p} I(S_l) + \frac{N_r}{N_p} I(S_r) \right\}. \tag{3.2.1}$$

The subscripts p, l, and r in the equations represent the parent node, the left child node, and the right child node, respectively. S denotes the dataset assigned to each node, and N represents the size of that dataset. The function I, which which measures the impurity of a node, uses the Gini index for classification as follows:

$$I_{Gini}(S) = 1 - \sum_{c=1}^{C} P(c|S)^2, \tag{3.2.2}$$

where C is the set of classes.

In the best case, each internal node partitions a training dataset into two balanced subsets, i.e., the root node splits $\tilde{N}$ data into two sets of $\tilde{N}/2$ data; its child nodes halves the data again, etc. Therefore, in this case, the complexity of the expected depth is $O(\log \tilde{N})$, and the complexities of the number of both leaf nodes and internal nodes become $O(\tilde{N})$. The computational complexity for training a tree is $O(K\tilde{N}(\log \tilde{N})^2)$, where Random Forest requires K searches for $\tilde{N}$ data for each $\log \tilde{N}$ depth because the sum of the numbers of node's data is $\tilde{N}$ at each depth, and the additional $\log \tilde{N}$ comes from the sorting procedure for optimization split thresholds [67, Sec. 5]. Table 3.2 lists each complexity when the size of the forest is M.

Extra Trees

Extra Trees is a further randomized decision tree ensemble [63]. While it is similar to Random Forest, it has two major differences. First, Extra Trees trains trees with N training data without bootstrapping, and second, it uses randomly selected split thresholds without optimization. It randomly selects several nodes and chooses the one that maximizes the value of Eq. 3.2.1. Therefore, Extra Trees can significantly reduce the computational complexity compared with Random Forest. In the best case, as shown in Table 3.2, each complexity, except for computational, is the same as that for Random Forest except using N instead of $\tilde{N}$. The computational complexity of Extra Trees for training M trees is $O(KMN \log N)$; because Extra Trees does not require any sorting procedure, there is no extra $\log N$ as there is in Random Forest [67, Sec. 5].

Random Ferns

Random Ferns is a fern ensemble specialized for image classification [60]. Its unique feature is the use of two attributes in each internal node instead of using an attribute–threshold pair. This comparison can be considered binary edge extraction in image processing:

$$f = \begin{cases} 1, & \text{if } I_{x,y} > I_{x',y'} \\ 0, & \text{otherwise} \end{cases}, \tag{3.2.3}$$

where $I_{x,y}$ and $I_{x',y'}$ are pixel intensities at the (x, y) and (x', y') coordinates, respectively.

Given images from a set of C classes, $C = \{1, 2, \ldots, C\}$, and a set of outcomes from a fern with D depths, $\mathcal{F} = \{f_1, f_2, \ldots, f_D\}$, Random Ferns finds the class label c^* by calculating

$$c^* = \arg\max_{c \in C} P(c|f_1, f_2, \ldots, f_D), \tag{3.2.4}$$

where (3.2.4) can be rewritten by Bayes' theorem as follows:

$$c^* = \arg\max_{c \in C} P(c)P(f_1, f_2, \ldots, f_D|c). \tag{3.2.5}$$

When using M ferns, Random Ferns assumes semi-naive Bayes. Let the outcome set of the m-th fern $\mathcal{F}_m = \{f_{m1}, f_{m2}, \ldots, f_{mD}\}$. Random Ferns considers all $\mathcal{F}_m$ as independent from each other but $f_{md} \in \mathcal{F}_m$ depends on other outcomes within $\mathcal{F}_m$. This assumption leads to

$$c^* = \arg\max_{c \in C} P(c) \prod_{m=1}^{M} P(\mathcal{F}_m|c). \tag{3.2.6}$$

Therefore, Random Ferns learns the probability distribution $P(\mathcal{F}_m|c)$ in its training phase.

The number of leaf nodes within a fern, L, is 2^D. Because this size is much larger than the size of the training data in many cases, there are many leaf nodes without assigned training data. A zero probability is not valid for (3.2.6) to get the proper class label. Therefore, Random Ferns assumes a Dirichlet prior to give some baseline probability to all possible outcome sets:

$$P(\mathcal{F}_m = l|c) = \frac{N_{lc} + \epsilon}{\sum_{i=1}^{L} (N_{ic} + \epsilon)}, \tag{3.2.7}$$

where N_{lc} is the number of c-class data assigned to the l-th leaf node, and ϵ is a small positive value: $\epsilon = 1$ in [60]. Note that the binary expression of l is identical to the outcome set of $\mathcal{F}_m$.

Table 3.2 shows each complexity of Random Ferns with M ferns. The most significant difference between Random Forest, Extra Trees, and Random Ferns is that the complexities of memory access and computation are much smaller in random ferns, with a complexity of $O(DMN)$.

rFerns

rFerns is an extension of Random Ferns for general-purpose classification [64]. It generates ferns with randomly selected pairs of attributes and thresholds, similar to Extra Trees. Because rFerns uses $\tilde{N}$ bootstrapped training data instead of N, the complexities of its memory access and computation are $O(DM\tilde{N})$, which are less than those of Random Ferns.

Summary

As shown in Table 3.2, these methods listed above have no significant differences from each other in the amount of computation and the number of memory accesses. Most of the calculations, except for Random Forest, consist of additions and comparisons, requiring one comparison operation and several addition operations to decide each branch of a decision tree from each data read. Assuming 32-bit data, the power consumption of one DRAM access is 6400x more than that of one integer addition [68]. Even assuming the number of addition operations is 10-100x larger than the number of DRAM accesses, its power consumption is still far less than that of the DRAM access.

Most calculations of Random Forest also consist of additions and comparisons but, in addition to them, it calculates the branching score multiple times proportional to the number of data read to find the optimal branching point. Each branching point calculation requires at least two multiplication for both left and right child nodes' impurity calculations. Although multiplication requires more power than addition, one DRAM access consumes 173x more energy than floating-point multiplication [68]. Even for two multiplications, the power consumption ratio is 86.5x, and memory access is still dominant for the power consumption in this case.

In these conventional methods, Random Forest and Extra Trees show better accuracy performance than others but require more memory accesses. That is why this study mainly focuses on memory access when designing a new decision tree ensemble algorithm in this paper.

3.3 Extra Ferns

Extra Ferns is an ensemble learning method based on Extra Trees and Random Ferns. This section provides an overview of Extra Ferns, details its TWO distinct key components, and analyzes its complexities.

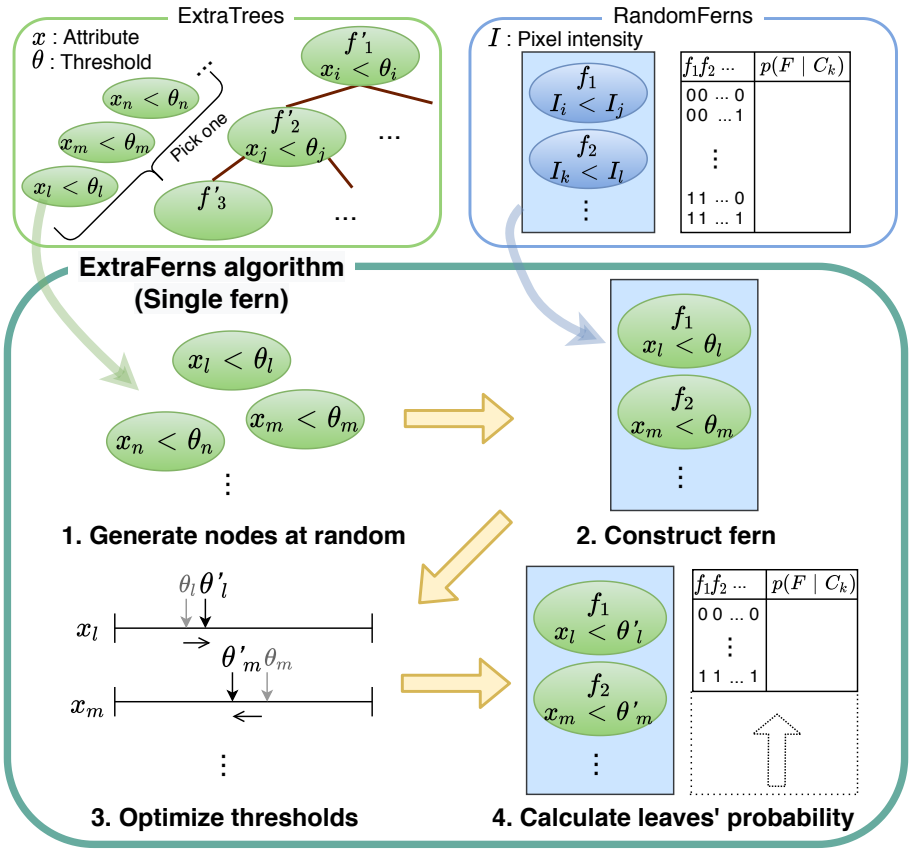

Fig. 3.2: Overview of Extra Ferns. Extra Ferns consists of random node generation, fern construction, parallel threshold optimization, and leaf probability calculation. The former two are based on conventional methods and bring memory access and computation advantages to Extra Ferns. The latter two enable Extra Ferns to achieve comparable performance to other methods.

3.3.1 Algorithm Overview

Fig. 3.2 shows an overview of Extra Ferns. The learning phase consists of four processes: 1. random node generation, 2. fern construction, 3. parallel threshold optimization, and 4. leaf probability calculation. The first two processes are based on conventional methods, and the other two processes are novel approaches devised for Extra Ferns. Extra Ferns builds a fern ensemble by repeating these four processes.

In random node generation, Extra Ferns generates nodes by randomly selecting pairs of attributes and thresholds like Extra Trees. Then, Extra Ferns builds a fern in the same manner as Random Ferns. By adopting these processes, Extra Ferns inherits the

Algorithm 1 Parallel Threshold Optimization

Input: Training dataset $\mathcal{T}$ & fern $\mathcal{F}$

Output: Tuple Θ^* of optimal thresholds for $\mathcal{F}$

$\Theta := (\theta_1, \theta_2, \ldots, \theta_D)$ where $\theta_d \in [0, 255]$ for $d \in \{1, 2, \ldots, D\}$
for each $u \in [1, U]$ **do**
  for each $d \in [1, D]$ **do**
    $\Theta' := \Theta$
    $G_{max} := 0$
    for each $r \in [-R, +R]$ **do**
      $\theta'_d := \theta_d + r$
      if $0 \leq \theta'_d \leq 255$ **then**
        $\Theta'(d) := \theta'_d$
        if $G_{max} < G(\Theta'|\mathcal{T}, \mathcal{F})$ **then**
          $G_{max} := G(\Theta'|\mathcal{T}, \mathcal{F})$
          $\theta^*_d := \theta'_d$
  $\Theta := (\theta^*_1, \theta^*_2, \ldots, \theta^*_D)$
$\Theta^* := \Theta$

} in parallel

characteristics of low memory access and low computational complexity from Extra Trees and Random Ferns. However, it is not a panacea because this randomized generation makes it difficult to achieve high inference accuracy: rFerns is a good example; it also adopts these processes, and its inference accuracy is significantly lower than those of other methods.

To solve this problem, Extra Ferns conducts parallel threshold optimization. Instead of the greedy search for optimizing each node's prediction performance, Extra Ferns searches the best thresholds for optimizing the entire fern's prediction performance. After that, Extra Ferns calculates each leaf node's probability distribution similarly to Random Ferns and rFerns. The distinct difference is that Extra Ferns does not use Dirichlet priors, allowing a more than 80% reduction in memory requirements for storing leaf nodes. The following two subsections describe the details of the parallel threshold optimization and the calculation of class probabilities for each leaf node, respectively.

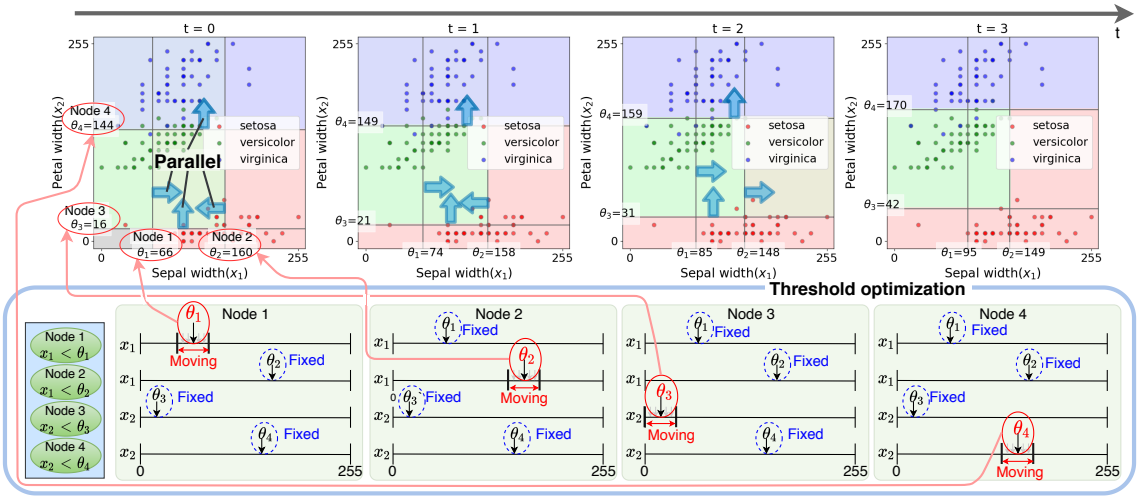

Fig. 3.3: Example of the parallel threshold optimization. The top row illustrates the concept of searching thresholds in parallel, and the bottom row shows the corresponding changes in the attribute space. Hyperparameters U, D, and R are set to 3, 4, and 11, respectively.

3.3.2 Parallel Threshold Optimization

Given a training dataset $\mathcal{T}$ and randomly generated fern $\mathcal{F}$, Extra Ferns employs the following objective function for threshold optimization:

$$G(\Theta|\mathcal{T},\mathcal{F}) = \max_{\Theta} \sum_{l=1}^{L} N_l(\mathcal{T}) \sum_{c=1}^{C} P(c|\mathcal{F}(\Theta) = l)^2, \tag{3.3.1}$$

where Θ represents the tuple of thresholds $\{\theta_1, \theta_2, \ldots, \theta_d\}$, N_l is the number of training data assigned to the l-th leaf, and $\mathcal{F}(\Theta)$ is the fern using Θ. To achieve non-greedy optimization, in (3.3.1), Extra Ferns computes the sum of each leaf's purities derived from Gini impurities, as show in (3.2.2). Then, Extra Ferns calculates the weighted sum of entire leaves' purities, where N_l is considered as a confidence factor.

For maximizing the objective function, Extra Ferns searches the best tuple of thresholds in parallel. Algorithm 1 shows the detailed process flow, where each training data is quantized with an 8-bit fixed-point expression in advance. As shown in Algorithm 1, the parallel threshold optimization has triple-nested loops. It executes the inner two loops for each update $u \in [1, U]$. Of the two inner loops, the outer loop corresponds to the node at each depth $d \in [1, D]$, and the innermost loop corresponds to the search range $r \in [-R, +R]$. Because the iterations of the two inner loops are mutually independent,

Extra Ferns executes them in parallel.

Fig. 3.3 shows an example of the parallel threshold optimization using the Iris dataset [69], where hyperparameters U, D, and R is set to 3, 4, and 11, respectively. The top row illustrates the concept of searching thresholds in parallel, and the bottom row shows the corresponding changes in the attribute space.

3.3.3 Sparsified Probability Distribution of Leaf Nodes

The basic equation for Extra Ferns is equivalent to (3.2.6) of Random Ferns. However, Extra Ferns take the logarithm of (3.2.6) because addition is more practically efficient than multiplication. Therefore, (3.2.6) can be rewritten as

$$c^* = \arg\max_{c \in C} \sum_{m=1}^{M} \log P(\mathcal{F}_m | c) + \log P(c). \tag{3.3.2}$$

The most significant difference is in how to handle the leaves without assigned training data. While Random Ferns uses Dirichlet priors, this requires a large amount of memory space for storing leaf node information. Extra Ferns reduce the memory space by modifying (3.2.7) to

$$P(\mathcal{F}_m = l | c) = \begin{cases} \frac{N_{lc}}{\sum_{i=1}^{L} N_{ic}} & (N_l \neq 0) \\ 1 & (N_l = 0) \end{cases}, \tag{3.3.3}$$

where N_l is the number of all data in the l-th leaf node. In (3.3.3), Extra Ferns equally assign 100% probability to every class when a leaf node is empty, ignoring the decision of an empty leaf node. Because the logarithm of one is zero, leaf information of Extra Ferns becomes sparse.

3.3.4 Complexity of Extra Ferns

Referring to Table 3.2, while Extra Ferns has the same complexities in leaf nodes and internal nodes as Random Ferns and rFerns, its number of non-empty leaf nodes is much smaller compared with Random Ferns and rFerns thanks to the sparsified leaf information. Due to the newly-introduced hyperparameters U and R, memory access and computational complexities of Extra Ferns are slightly larger than those of conventional methods. Because R is usually a small constant value from 5 to 10, the analysis presented here exclude R from the computational complexity, defining it as $O(UDMN \log N)$. The additional $\log N$ comes from the process of storing sparsified leaf nodes.

TABLE 3.4: Evaluation Dataset Description.

Dataset	# Training Data	# Test Data	# Attributes	# Classes
MNIST	60,000	10,000	784	10
Fashion-MNIST [70]	60,000	10,000	784	10
Kuzushiji-MNIST [71]	60,000	10,000	784	10
Devanagari-Script [72]	76,666	15,334	1,024	46
HIGGS [73]	80,000	20,000	28	2
SUSY [73]	80,000	20,000	18	2

The computational complexity of Extra Ferns is not much different from that of Random Forest and Extra Trees, and the training time is about the same as for conventional decision tree ensembles, e.g., a few seconds or minutes to run on the CPU. This time is much less than the training time for deep neural networks, which requires a few hours or days to run on the GPU, which is one of the reasons why decision tree ensembles are suitable for training on edge.

3.4 Experiments

This section begins by identifying the appropriate hyperparameters for Extra Ferns through preliminary experiments and subsequently evaluates the performance of Extra Ferns.

3.4.1 Preliminary Experiments

Before evaluation, it is necessary to determine appropriate hyperparameters in advance because they impact the threshold optimization results. It is also important to verify how the absence of both bootstrapping and the Dirichlet prior influences inference accuracy. This section conducted a grid search of hyperparameters and examined the accuracy of Extra Ferns by adding bootstrapping and the Dirichlet prior. Table 3.4 describes the datasets used in this study, where HIGGS and SUSY are randomly subsampled datasets of the originals.

Hyperparameter Grid Search

As mentioned above, Extra Ferns newly introduces two hyperparameters R and U, where R is the range of the threshold search, and U is the number of updates. If either R or U is

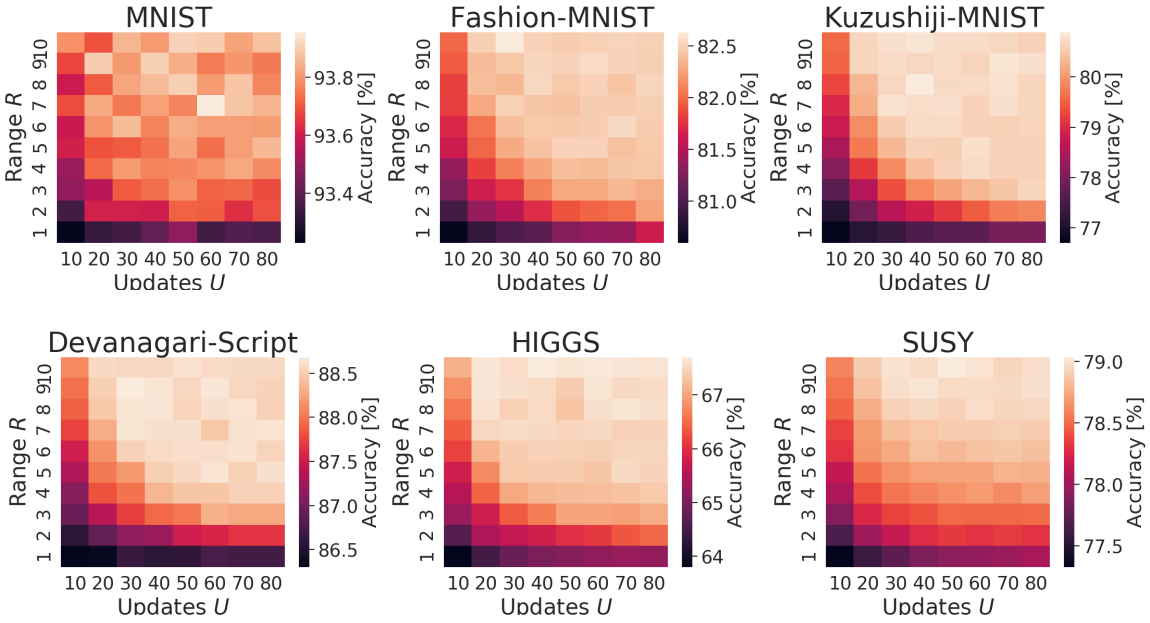

Fig. 3.4: Heatmaps of the hyperparameter grid search. Two hyperparameters R and U are examined on datasets in Table 3.4, and each heatmap shows a similar tendency. Based on the results, this work uses $R = 8$ and $U = 30$ as the parameters for other evaluation.

too small, Extra Ferns falls into one of the local optimal solutions. To find appropriate R and U, this work conducted a grid search on datasets in Table 3.4. Fig. 3.4 shows the grid search results for accuracy in the form of heatmaps, where M and D are set to 500 and 15, respectively. As shown in Fig. 3.4, each heatmap shows a similar tendency, and based on these results, this work sets R and U to 8 and 30, respectively.

Influences of Bootstrapping and Dirichlet Prior

This study designed Extra Ferns without bootstrapping and Dirichlet priors based on empirical results. Table 3.5 lists the results comparing the baseline mode and other variations with bootstrapping and Dirichlet priors, where M and D are set to 1,000 and 15, respectively. From Table 3.5, it is evident that the baseline outperforms both bootstrapping variation and Dirichlet variation except in one case. Based on these results, this work excluded both processes from Extra Ferns.

TABLE 3.5: Preliminary Evaluation of Extra Ferns Variations

Dataset	Accuracy[%] (diff)			
	Extra Ferns baseline	Extra Ferns + bootstrap	Extra Ferns + Dirichlet $\epsilon = 0.1$	Extra Ferns + Dirichlet $\epsilon = 1$
MNIST	**93.9**	93.5 (−0.4)	92.7 (−1.2)	92.0 (−1.9)
Fashion-MNIST	**82.5**	80.6 (−1.9)	80.8 (−1.7)	80.2 (−2.3)
Kuzushiji-MNIST	**80.9**	78.8 (−2.1)	77.1 (−3.8)	74.5 (−6.4)
Devanagari-Script	**88.9**	86.4 (−2.5)	85.2 (−3.7)	82.1 (−6.8)
HIGGS	68.0	66.7 (−1.3)	**68.1** (+0.1)	**68.1** (+0.1)
SUSY	**78.9**	78.7 (−0.2)	78.8 (−0.1)	78.7 (−0.2)

3.4.2 Evaluation

This section compared Extra Ferns with conventional decision tree ensembles in terms of accuracy, memory space, memory access, and power consumption where Random Forest, Extra Trees, and rFerns were comparison targets; The analysis excluded Random Ferns because it is not applicable to general tabular data. Each implementation came from machine learning libraries: Random Forest and Extra Trees from scikit-learn (v0.22.1) and rFerns (v3.0.0) from R package. In the experiments, the number of trees/ferns is 1,000, and other values are their default. Each reported result is an average of ten trials.

Accuracy

Table 3.6 lists the results for accuracy of each method, where D is 15, 20, and 25. However, this work measured the accuracy of rFerns only for D set to 15 because the implementation in the R package of rFerns only allows a depth of up to 17. It seems that this is due to the large memory requirement of rFerns because it stores all 2^D leaf nodes.

First, when focusing on each method at depth 15, the average accuracy of Extra Ferns was 9.3% higher than that of rFerns thanks to the threshold optimization. However, it was 3.0% and 1.2% lower than those of Random Forest and Extra Trees, respectively, due to the model simplicity.

Second, the comparison focuses on each method at the accuracies for the D with the highest accuracy. The shaded areas in the table are the highest accuracy for each method. Extra Ferns significantly outperformed rFerns, except on one dataset. In two of the six datasets, Extra Ferns was at least 20% more accurate than rFerns. Extra Ferns was inferior

TABLE 3.6: Accuracy Comparison Between Ensemble Methods.

Dataset	D (Depth)	Accuracy[%] (diff)			
		Extra Ferns	Random Forest	Extra Trees	rFerns
MNIST	15	93.9	96.8	96.7	**86.9** (−8.9)
	20	95.2	**97.1** (+1.3)	97.3	N/A
	25	**95.8**	**97.1** (+1.3)	**97.4** (+1.6)	
Fashion MNIST	15	82.5	87.1	86.3	**74.2** (−10.2)
	20	83.9	87.7	87.4	N/A
	25	**84.4**	**87.9** (+3.5)	**87.7** (+3.3)	
Kuzushiji MNIST	15	80.9	84.9	82.3	**59.3** (−24.9)
	20	83.4	86.4	86.1	N/A
	25	**84.2**	**86.5** (+2.3)	**87.1** (+2.9)	
Devanagari Script	15	88.9	89.6	87.5	**70.1** (−20.0)
	20	90.0	92.0	92.0	N/A
	25	**90.1**	**92.2** (+2.1)	**92.6** (+2.5)	
HIGGS	15	**68.0**	72.5	68.5	**68.8** (+0.8)
	20	67.2	**72.8** (+4.8)	70.3	N/A
	25	66.8	**72.8** (+4.8)	**71.1** (+3.1)	
SUSY	15	78.9	**80.0** (+0.8)	79.2	**78.1** (−1.1)
	20	**79.2**	**80.0** (+0.8)	79.7	N/A
	25	79.1	**80.0** (+0.8)	**80.0** (+0.8)	

to Random Forest and Extra Trees in all cases. However, the accuracy difference was always within 5%, and the average differences were 2.5% and 2.4% for Random Forest and Extra Trees, respectively.

Memory Space

For comparison, this study considers only the dominant factors occupying memory space in each method: internal nodes and leaf nodes. Table 3.7 lists the number of leaf nodes per tree for each method. Shaded areas indicate the highest accuracy of each method in Table 3.6. In Table 3.7, Extra Ferns shows smaller leaf node sizes than rFerns except for

TABLE 3.7: Comparison of Average Numbers of Leaf Nodes per Tree.

Dataset	D (Depth)	# Leaf Nodes L			
		Extra Ferns	Random Forest	Extra Trees	rFerns
MNIST	15	539	3,803	5,590	32,768
	20	1,701	4,799	9,205	N/A
	25	4,356	4,942	10,041	
Fashion MNIST	15	3,010	2,599	4,378	32,768
	20	8,730	4,237	8,714	N/A
	25	16,042	4,771	10,723	
Kuzushiji MNIST	15	6,520	4,421	5,371	32,768
	20	18,387	6,460	10,517	N/A
	25	30,684	6,754	12,733	
Devanagari Script	15	3,371	7,733	10,007	32,768
	20	15,666	14,615	23,690	N/A
	25	35,510	15,931	28,500	
HIGGS	15	5,337	3,884	2,935	32,768
	20	26,844	8,393	10,348	N/A
	25	52,805	10,623	21,011	
SUSY	15	1,388	2,818	2,292	32,768
	20	6,605	5,921	7,969	N/A
	25	16,089	7,990	16,602	

Devanagari-Script because there is no need to consider Dirichlet priors with Extra Ferns. Compared with Random Forest and Extra Trees, however, more leaf nodes were required in most cases.

This work estimates the memory space requirements for each method based on Tables 3.8 and 3.9. Table 3.8 lists the formulae to calculate the memory sizes for internal and leaf nodes, and Table 3.9 lists the byte size of each constant. For internal nodes, each node includes a pair of an S_x-byte attribute x and an $S_{\theta_{f/i}}$-byte threshold θ. In contrast to other methods using 4-byte float thresholds, Extra Ferns adopts 1-byte integer thresholds. Also, Random Forest and Extra Trees require an S_{ptr}-byte pointer to each child node. For

TABLE 3.8: Memory Space Requirements for Internal and Leaf Nodes.

Algorithm	Internal Nodes [byte]	Leaf Nodes [byte]
Random Forest Extra Trees	$(L-1)(S_x + S_{\theta_f} + S_{\text{ptr}})$	LCS_P
rFerns	$D(S_x + S_{\theta_f})$	
Extra Ferns	$D(S_x + S_{\theta_i})$	$L(CS_P + 2S_{\text{ptr}} + S_{\text{index}})$

* refer to Table 3.7 for L

TABLE 3.9: Byte Size of Each Constant.

Constant	Description	Byte
S_x	size of attribute	2
S_{θ_f}	size of float threshold	4
S_{θ_i}	size of integer threshold	1
S_{ptr}	size of node address	2
S_{index}	size of index	2 or 4
S_P	size of leaf's probabilities	4

leaf nodes, each node includes a probability distribution for each class. Because Extra Ferns' implementation stores sparsified leaf nodes in an associative container, its leaf nodes include two S_{ptr}-byte pointers and an S_{index}-byte index. When the leaf node size is less than or equal to 2^{16}, this analysis uses a 2-byte integer for each index. When it is more than 2^{16}, this analysis uses a 4-byte integer instead.

Fig. 3.5 depicts the data sizes of internal and leaf nodes per tree when D is 15 or 25. Results of rFerns are excluded from all graphs because the number of leaf nodes of rFerns is too large to place on the graphs. As shown in Fig. 3.5, while Extra Ferns requires a smaller or equivalent memory size for depth 15 compared with conventional methods, it requires more memory on average for depth 25. This large memory requirement is a weakness of the current Extra Ferns.

Memory Access

This work estimates the number of fetched training data from off-chip memory as off-chip memory accesses for the training of each method based on the following three cases: 1.

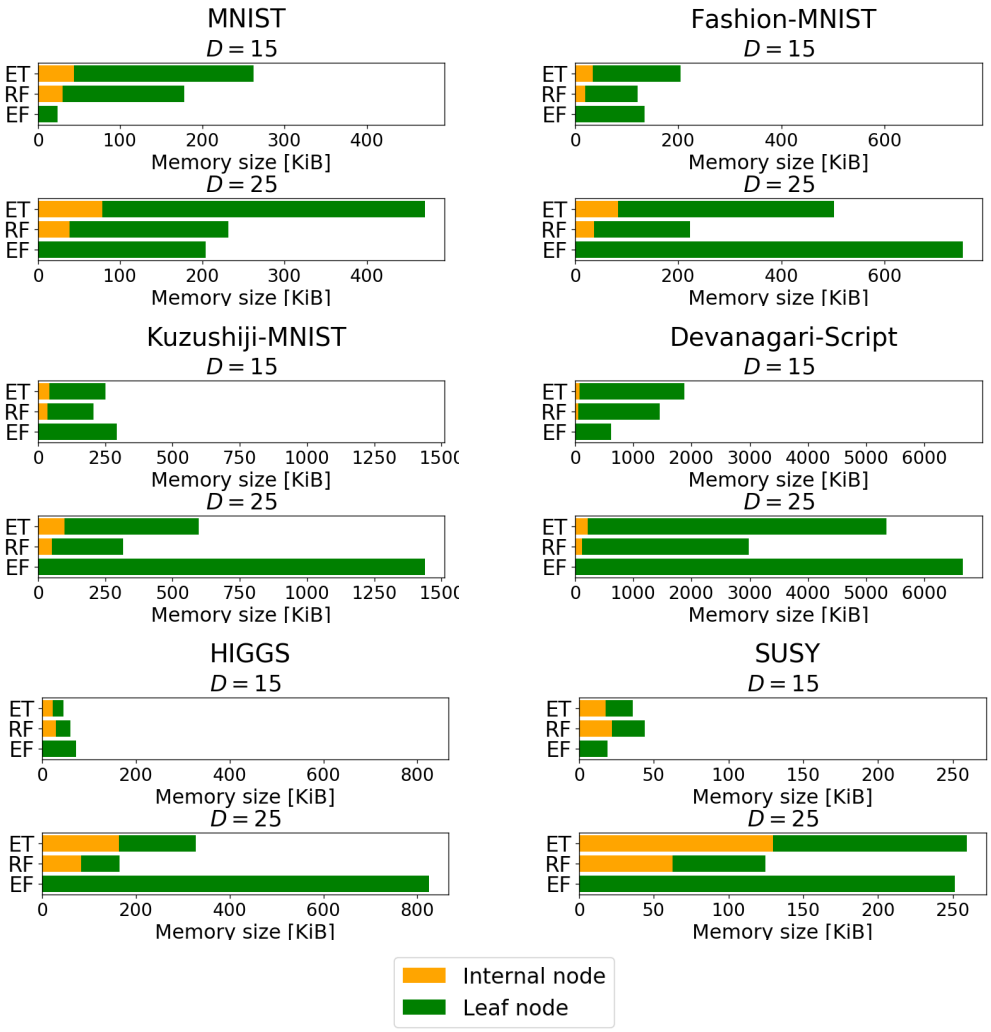

Fig. 3.5: Memory requirement breakdown of each ensemble method. ET, RF, and EF represent Extra Trees, Random Forest, and Extra Ferns, respectively. When the depth is 15, Extra Ferns requires an equivalent or smaller memory size to other methods, but when the depth is 25, Extra Ferns requires more memory space than others due to the large number of leaf nodes.

small N (number of training data) and few attributes, 2. small N and many attributes, and 3. large N.

1. Small N and Few Attributes

This case corresponds to HIGGS and SUSY in Table 3.4. Because it is possible to put

all the training data in the on-chip memory, the number of off-chip memory accesses is equivalent between all methods by storing the data in on-chip memory at the beginning of the training.

2. Small N and Many Attributes

This case corresponds to all datasets except HIGGS and SUSY in Table 3.4. In this case, this work estimates that the number of fetched data from off-chip memory for Extra Ferns and rFerns is DN per tree because each of D nodes has one splitting attribute. Extra Ferns must store the DN samples in on-chip memory and use them every time it updates thresholds. Additionally, this study estimates that the number of fetched data from off-chip memory for Random Forest and Extra Trees is $KN \log L$ per tree because, in the best splitting case, the depth is $\log L$, and each node has K different candidate attributes (see Section 3.2). Also, Random Forest and Extra Trees execute random access because the subset of training data changes for each update, except for the root node.

The upper half of Table 3.10 gives the ratio of fetched training data from off-chip memory for each method when D is 15. Random Forest and Extra Trees required 23.4 and 24.3 times more training data from off-chip memory than Extra Ferns, respectively. Therefore, Extra Ferns required only 4.3% and 4.1% off-chip memory accesses compared to Random Forest and Extra Trees, respectively.

3. Large N

This case corresponds to the original full datasets of HIGGS and SUSY. In this case, the number of fetched data from off-chip memory for Extra Ferns is UDN per tree because it is impossible to store DN samples in on-chip memory. Although Extra Ferns' threshold optimization requires U times more off-chip memory accesses than rFerns, the increase in accuracy far outweighs this disadvantage. In Random Forest and Extra Trees, each node must read data twice for node selection and data division because each node cannot store all data used in node selection due to too much data. Because the number of nodes is $O(N)$ from Table 3.3, this work assumes that the tree makes nodes up to max depth D, and the number of fetched data from off-chip memory is $2KND$ per tree.

The lower half of Table 3.10 shows the ratio of training data amount fetched from off-chip memory for original HIGGS and SUSY. For comparison, this analysis assumes the same D for the four methods. The ratio of Extra Ferns is multiplied by U set to 30 in this paper. Both ratios of Random Forest and Extra Trees multiplied by $2K$, where this analysis sets 5 and 4 to K for HIGGS and SUSY, respectively. As a result, Extra Ferns requires 3 times and 3.75 times more data fetches for HIGGS and SUSY than Random Forest and Extra

TABLE 3.10: Ratio of Fetched Training Data from Off-Chip Memory.

Case	Dataset	Extra Ferns	rFerns	Random Forest	Extra Trees
2	MNIST	**15**	**15**	333 (×22.2)	349 (×23.2)
	Fashion-MNIST	**15**	**15**	318 (×21.2)	339 (×22.6)
	Kuzushiji-MNIST	**15**	**15**	339 (×22.6)	347 (×23.1)
	Devanagari-Script	**15**	**15**	413 (×27.6)	425 (×28.3)
	Average	**15**	**15**	351 (×23.4)	365 (×24.3)
3	Original HIGGS	30	1	10	10
	Original SUSY	30	1	8	8
	Average	30	1	9	9

Trees need. Because Random Forest and Extra Trees require random access to off-chip memory, it is not always true that Extra Ferns is inferior to Extra Trees or Random Forest. Still, it is undeniable that Extra Ferns has a weak point against large-size datasets and needs to be improved.

Power Consumption

It is possible to estimate the power consumption of Extra Ferns by multiplying the number of operations and the energy consumption for each operation. As mentioned in Section 3.2, however, the off-chip memory access operation takes 6400x more energy than the addition or the comparison operation needs. So even if each method require 10-100x more computational operations than off-chip memory access, off-chip memory access is still the most dominant factor in power consumption. This case can happen only when the training data size from off-chip memory is small enough to store all of them in on-chip memory. So, in usual, the power consumption for computation is negligible in Extra Ferns, so that this work approximately estimates the total power consumption by considering only the off-chip memory access.

Fig. 3.6 shows the estimated power consumption per tree for four datasets: MNIST, Fashion-MNIST, Kuzushiji-MNIST, and Devanagari-Script. From [68], this analysis assumes DRAM access consumes 20 pJ/bit. In each graph, red bars and blue bars show the power consumption required to load training data from DRAM and write a trained model back to it. At a depth of 25, this work excluded rFerns from evaluation targets because its model size is too large to store. In Fig. 3.6, Extra Ferns shows the minimal power

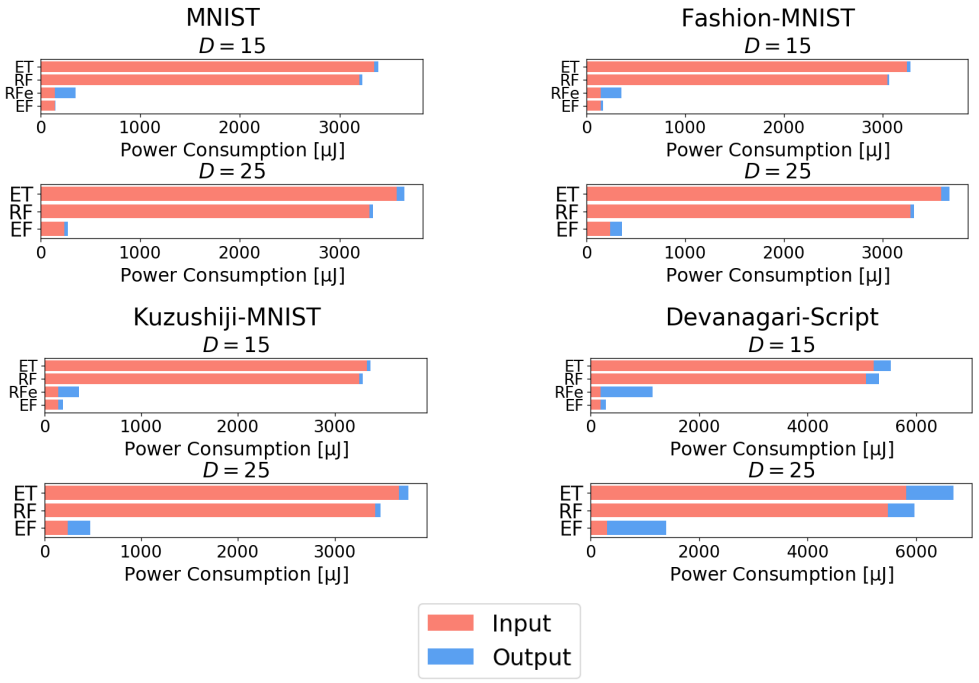

Fig. 3.6: Power consumption comparison between MNIST, Fashion-MNIST, Kuzushiji-MNIST, Devanagari-Script. ET, RF, RFe, and EF represent Extra Trees, Random Forest, rFerns, and Extra Ferns. In each graph, red bars and blue bars show the power consumption required to load training data from DRAM and write a trained model back to it. The results show Extra Ferns requires minimal power consumption in every condition.

consumption for all four datasets thanks to the lightweight data read requirement. However, as mentioned in Section 3.4.2, the model size becomes more significant at a depth of 25. For two of the four datasets, the model size becomes larger than the amount of training data read, resulting in increased power consumption. The model size with large depths is a problem even in terms of power consumption.

3.5 Extra Ferns with Random Projection

As mentioned in previous section, Extra Ferns achieves better memory access efficiency but still has room for improvement in memory usage. This section shows that random projection (RP) [65, 66] improves the accuracy of Extra Ferns with a low depth, especially on image datasets, resulting in memory usage reduction. This is not the ultimate solution for the problem but a piece of evidence that there still exist ways to enhance Extra Ferns.

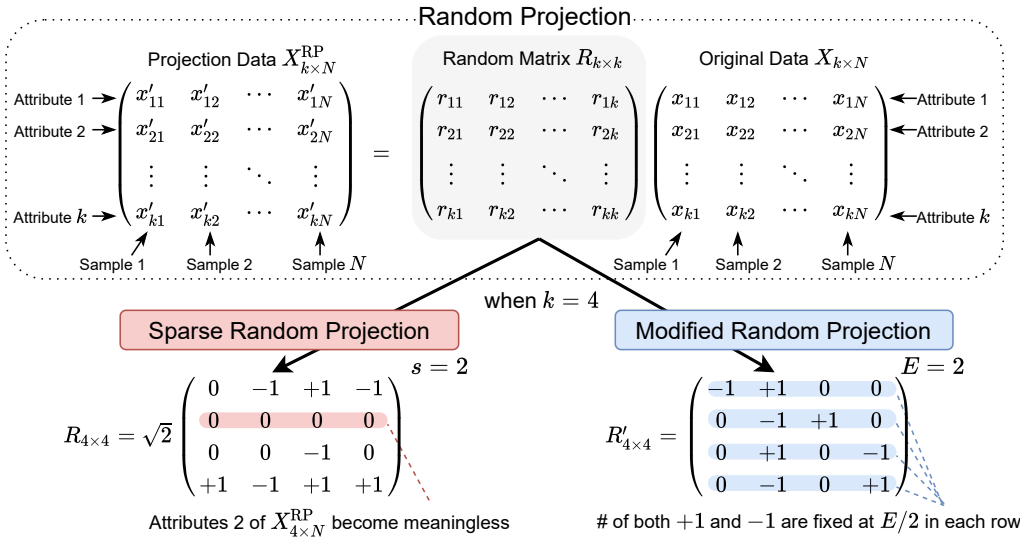

Fig. 3.7: Overview of RP for Extra Ferns. The k-dimensional original data with N samples $\boldsymbol{X}_{k\times N}$ are projected to $X^{\mathrm{RP}}_{k\times N}$ using a random $k \times k$-dimensional matrix $\boldsymbol{R}_{k\times k}$. The dimensionalities of the original data and the projection data are the same. The modified RP is a variation of sparse RP with a restriction on the number of non-zero values in each row vector and without the multiplication of $\sqrt{s}$ because quantization follows RP in Extra Ferns.

3.5.1 Modified Random Projection for Extra Ferns

RP is a linear transform method usually used for dimensionality reduction [65, 66]. This study proposes applying a modified RP to Extra Ferns as a preprocessing step, which achieves higher accuracy. Fig. 3.7 shows the difference between the original RP and this work's modified RP. As shown in Fig. 3.7, given the original data matrix X, RP linearly transforms the original data X to the projection data X^{RP} with the random matrix R. The random matrix R is not necessarily square, but this work uses a square matrix $R_{k\times k}$ to preserve the original data's dimensionality. Also, sparse RP [66] stochastically defines non-zero r_{ij} as

$$r_{ij} = \begin{cases} +\sqrt{s} & \text{with probability } \frac{1}{2s} \\ 0 & \text{with probability } 1 - \frac{1}{s} \\ -\sqrt{s} & \text{with probability } \frac{1}{2s} \end{cases}, \tag{3.5.1}$$

where the hyperparameter s is usually a large number proportional to the number of attributes. However, modified RP leaves a constant number of non-zero r_{ij} in each row, where this constant is E in Fig. 3.7. This simple modification allows better performance

TABLE 3.11: Comparison of Extra Ferns accuracy with and without RP.

Dataset	Accuracy[%]			
	Depth 15		Depth 25	
	w/o RP	w/ RP	w/o RP	w/ RP
MNIST	93.9	95.9 (+2.0)	95.8	96.2 (+0.4)
Fashion-MNIST	82.5	83.7 (+1.2)	84.4	84.3 (-0.1)
Kuzushiji-MNIST	80.9	82.1 (+1.2)	84.2	83.8 (-0.4)
Devanagari-Script	88.9	89.0 (+0.1)	90.1	87.9 (-2.2)
HIGGS	68.0	63.3 (-4.7)	66.8	62.5 (-4.3)
SUSY	78.9	77.8 (-1.1)	79.1	78.5 (-0.6)

with Extra Ferns.

3.5.2 Effectiveness of Modified Random Projection

Fig. 3.8 and Fig. 3.9 show accuracy comparison results between bare Extra Ferns, Extra Ferns with the original RP, and Extra Ferns with the modified RP. The comparison is conducted on two configurations, $D = 15$ and $D = 25$, and six datasets: MNIST, Fashion-MNIST, Kuzushiji-MNIST, Devanagari-Script, HIGGS, and SUSY. In each comparison, this analysis investigated accuracies with E in the range of 2 to 10. Also, the analysis set s as half of the number of attributes for $E = 2$, one third for $E = 3$, etc. for fair comparison. Each row of Table 3.11 compares the Extra Ferns' accuracies with and without RP, where the accuracy with RP is the maximum value of each corresponding graph shown in Fig. 3.8. The blue indicates the higher accuracy, and the red indicates the lower accuracy at the same depth.

As shown in Fig. 3.8 and Table 3.11, the modified RP shows better performance than the original RP, improving the accuracy by up to 2.0% and 1.1% on average at $D = 15$ and $E = 2$, respectively for the four datasets: MNIST, Fashion-MNIST, Kuzushiji-MNIST, and Devanagari-Script. For the other two datasets, HIGGS and SUSY, both Extra Ferns with RP showed accuracy drops under all configurations, dropping by 2.9% on average at depth 15, as shown in Fig. 3.9. The most significant difference between these two groups of datasets is their data type, i.e., the modified RP has good compatibility with the former four datasets because they contain image data with correlated attributes. However, when the depth is 25, RP is ineffective for three out of four image datasets, decreasing the accuracy by 0.9% on

TABLE 3.12: Accuracy Comparison between Extra Ferns with RP and Extra Ferns with Random Ferns at Different Ratios.

Dataset	Accuracy[%]					
	+Random Ferns' nodes					+Random projection
	Ratio of Random Ferns' nodes					
	0	0.25	0.5	0.75	1	
MNIST	93.9	**94.0**	93.8	93.8	93.8	95.9
Fashion-MNIST	82.5	**82.6**	82.4	82.5	82.3	83.7
Kuzushiji-MNIST	**80.9**	80.7	80.3	79.8	79.3	82.1
Devanagari-Script	**88.9**	88.7	88.5	88.1	87.8	89.0

average. Therefore, RP is only effective for the image datasets when the depth of Extra Ferns is shallow. This means Extra Ferns can improve accuracy for image datasets by using the modified RP instead of deepening the fern structure, saving significant memory usage.

3.5.3 Extra Ferns with Random Projection and Random Ferns

Extra Ferns with the modified RP includes Random Ferns as a subset; when $E = 2$ and the threshold in each node is 0, the structure of Extra Ferns is identical to that of Random Ferns. Therefore, it would be an interesting analysis to compare Extra Ferns and Random Ferns and investigate what happens when merging the two methods. The merging process combines Extra Ferns and Random Ferns in specific ratios, and its performance is evaluated on image datasets, as Random Ferns is not applicable to tabular datasets.

Table 3.12 lists the accuracy comparison results at $D = 15$ on the four image datasets. In Table 3.12, the column with a ratio of 0 represents bare Extra Ferns, and the column with a ratio of 1 represents bare Random Ferns. The middle columns list the accuracies of the methods merged with the corresponding ratios. Also, the right-most column represents Extra Ferns with RP. Table 3.12 indicates that Extra Ferns with RP achieves higher accuracy than other variations and is more effective than Random Ferns, even on image datasets.

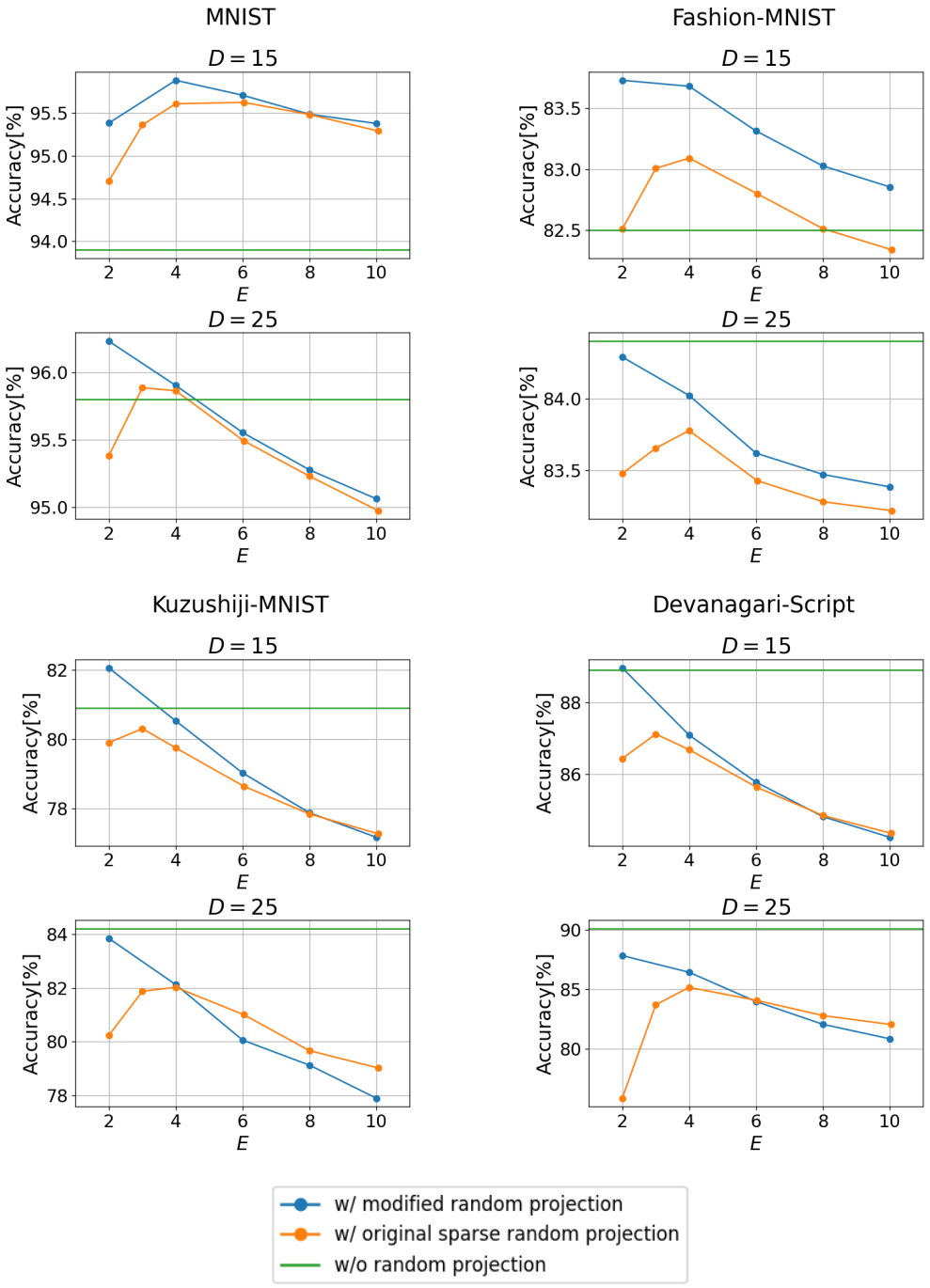

Fig. 3.8: Experimental results for the effect of RP for image datasets. The parameter E on the horizontal axes is the expected value of the number of non-zero values in each row of R. The RP was effective for datasets with large dimensions, and $D = 15$ was more effective than $D = 25$. In comparing the implementation of RP, the modified RP was more accurate for the datasets where RP was effective.

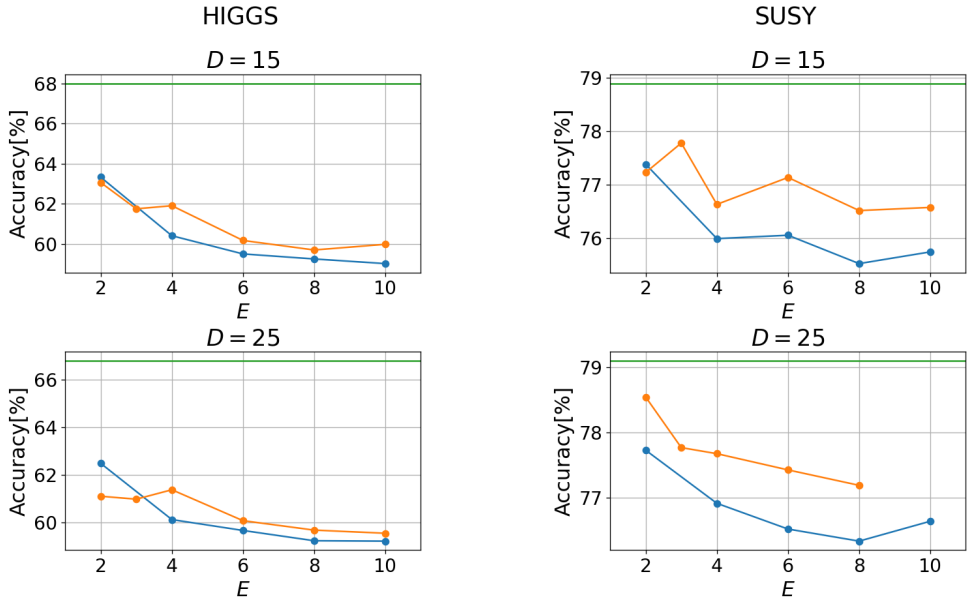

Fig. 3.9: Experimental results for the effect of RP for tabular datasets. The axis labels and legend colors are identical to those in Fig. 3.8. The RP was not effective for low dimensional data sets, resulting in lower accuracy.

3.6 Conclusion

This chapter proposed an edge-oriented decision tree ensemble called Extra Ferns that requires extremely low memory access for training. For achieving high inference accuracy, Extra Ferns conducts non-greedy optimization so that it can significantly outperform rFerns based on the fern structure. Experimental results show that Extra Ferns required only 4.3% and 4.1% memory access for training models only with 3.0% and 1.2% accuracy drops compared with Random Forest and Extra Trees, respectively. However, Extra Ferns has a weakness in memory usage, which increases significantly with its depth. To increase the accuracy without an increase in the depth, this work proposed applying modified RP to Extra Ferns and showed that RP improves Extra Ferns' accuracy by up to 2.0% on the image datasets, which have correlated attributes. The future work will aim to achieve further accuracy improvement and lower memory usage.

Chapter 4

Improving Collaborative Effectiveness in Ensemble-Based Inference at the Edge

This chapter examines methods to enhance the accuracy and efficiency of ensemble techniques for DL in edge environments. DL, known for its ability to achieve high accuracy on complex tasks, presents a promising approach for Edge Ensembles. However, compared to classical machine learning methods such as decision trees, DL entails higher computational costs and lower inference efficiency on individual devices. This study focuses on improving the overall computational efficiency of ensemble members by optimizing the accuracy and efficiency of the model integration component in DL ensembles.

4.1 Introduction

Edge-based inference of deep neural networks (DNNs) has garnered attention due to their ability to address challenges related to real-time performance, data privacy, and scalability, which are often associated with traditional cloud-based DNN inference [27]. In recent years, collaborative inference, involving the cooperative use of multiple edge nodes with individually limited computational resources to function as a larger computational resource, has been actively researched [28]. This approach overcomes the challenge of scarce computational resources in single nodes by leveraging the collective power of multiple nodes, resulting in more effective and accurate inference. This study focuses on the collaborative inference system among edge devices to efficiently process the expanding data volume resulting from the growing IoT ecosystem. The conventional collaborative inference architecture partitions a single DNN model and distributes its components across multiple edge devices [25, 27, 28]. While this architecture allows establishing large networks on the edge, it has certain drawbacks, such as the difficulty in conducting inference

on a single device, the strong dependency on availability of neighboring devices with required DNN partitions, and the need for complex control of repeated device-to-device communications [24].

To address these challenges, ensemble-based collaborative inference systems, Edge Ensembles, have been proposed [6, 42, 43, 74]. Edge Ensembles utilize ensemble methods to aggregate the inference results from respective models deployed on each edge device, achieving higher accuracy than single-device inference. This approach offers several distinct advantages that underscore its significance within the edge environment: 1) **Low dependence on other devices**: Edge Ensemble allows inference even when communication is disrupted because each device has a complete local model. 2) **Scalability**: The adaptability of Edge Ensembles allows for the adjustment of the number of devices involved based on the complexity of the task, enhancing scalability as demands fluctuate. 3) **Decentralization**: Operating within a decentralized system, Edge Ensembles reduce reliance on centralized servers, enhancing the robustness and reliability of edge-based inference. 4) **Resource Optimization**: Edge Ensembles make efficient use of computational resources by leveraging unused models on certain devices for inference, thus optimizing the utilization of available computing power. In particular, the last advantage aligns well with the explosive growth of IoT devices in the future. As IoT devices continue to increase, efficiently utilizing unused computational resources will become challenging. Edge ensembles offer a straightforward and efficient way to use these computational resources without the need for complex tasks such as network partitioning. Edge Ensembles share similarities with Federated Learning (FL) [21], regarding collaborative processing systems across multiple devices. However, it is important to note that Edge Ensemble is a collaborative inference system, whereas FL is a collaborative training system.

In Edge Ensembles, idle or low-utilization neighboring devices assist with inference tasks of other device. However, due to the higher computational demands of DNN models compared to traditional machine learning methods, the resource availability for inference task on other devices may be significantly reduced. Under these circumstances, improving the efficiency of model integration methods in edge environments offers a practical solution for enhancing the overall efficiency of Edge Ensemble clusters. By adopting efficient or highly accurate model integration techniques, the number of models required to achieve a given level of accuracy can be reduced. This reduction decreases the number of devices needed for the ensemble, thereby increasing the availability of devices across the cluster.

In previous Edge Ensembles, they have utilized conventional model integration methods such as majority voting [74], averaging the output of each model [6, 74], and weighted averaging based on the importance of each model [42, 43]. In addition to these methods,

various effective model integration techniques have been used in previous DNN ensembles [29, 75–77]. However, since DNN ensembles require high computational costs due to the processing of multiple DNN models, most of these techniques were not originally designed for edge inference systems. It is uncertain whether these previous integration techniques can be directly applied in the context of Edge Ensembles, where there are constraints on computational resources and uncertainty regarding the availability of other devices' models due to limited communication bandwidth.

Given these challenges, this study examines the model integration part for improving Edge Ensembles. The model integration techniques commonly employed in DNN ensembles can be categorized into two approaches. The first approach involves handling the entire process, from training base learners to integrating them in inference. The second approach focuses on integrating the individual models already trained in each method. An example of the former approach is the Mixture of Experts (MoE) [53], which consists of task-specific expert models and a gating function that selects these experts based on the type of input. In recent years, the MoE models have achieved significant high accuracy in computer vision and natural language processing [55, 78, 79]. However, the former approaches, like MoE, which train base learners with the assumption of ensembling, do not meet the requirements of Edge Ensembles, where each base learner can conduct inference independently.

Therefore, this work considers improving the model integration in Edge Ensembles based on the latter approach and selects three integration techniques used in previous DNN ensembles: cascade, weighted averaging, and test-time augmentation (TTA). It applies these techniques to models designed for Edge Ensembles and evaluates their effectiveness, while also introducing improvements and extensions to better align these methods with the requirements of Edge Ensembles. Cascades are techniques that infer models in sequence and terminate the inference process midway, which is less computationally expensive but has higher latency than simple ensembles. To tackle this latency challenge, this study proposes a method aimed at optimizing this aspect for edge scenarios. For TTA and weighted averaging, in addition to investigating the effects of these methods on Edge Ensembles, this study proposes learning TTA policies and the weights of weighted averaging using ensemble inference labels instead of ground truth labels. It is important to note that these methods focus on adapting the ensemble integration part during inference in edge environments with limited labeled data, rather than training the base learners themselves. The contributions of this chapter are summarized as follows:

1. This study analyzes the effectiveness of Edge Ensembles with three integrated methods: cascade, TTA, and weighted averaging, in terms of accuracy, computational costs, and latency.

2. This study proposes advancements from existing methods, including the m-parallel cascade, which allows for the adjustment of latency and computational complexity while maintaining accuracy, and ensemble label-based learning, which learn ensemble weights and TTA policies without ground truth labels.

3. The m-parallel cascade achieve a reduction in computational costs and latency while maintaining accuracy compared to the conventional Edge Ensemble.

4. The ensemble label-based learning demonstrates comparable performance to conventional training methods that use training data. This approach enables higher accuracy in model integration in edge environments using only inference data, eliminating the need for labeled training data.

The rest of this chapter is organized as follows. Section 4.2 describes Edge Ensembles and the ensemble methods used in this study. Section 4.3 describes the proposed methods, including m-parallel cascade and ensemble label-based learning. Section 4.4 first sets up the experiments, then analyzes the hyperparameters of the proposed methods, and finally presents the experimental results and their analysis. Finally, Section 4.5 concludes this chapter.

4.2 Related Work

This section first explains ensemble-based collaborative inference, Edge Ensembles, which is the focus of this study. Then, it describes conventional model integration techniques for improving ensemble methods and discuss how to use them in the Edge Ensembles context.

4.2.1 Edge Ensembles

Edge Ensembles are collaborative inference systems that improve accuracy by aggregating individual models' predictions on each edge device. Fig. 4.1 represents an overview of the system. The figure illustrates the collaborative inference flow in Edge Ensembles. In this process, the local device shares the collected data with other nearby devices. Each device conducts inference using the shared data and sends the results back to the local device. Finally, the local device aggregates all the inference results, including its own output.

Edge Ensembles offer several advantages in the following aspects: they enable standalone inference using a local model, provide scalability by adjusting the number of devices involved based on task complexity, and operate as decentralized systems without dependency

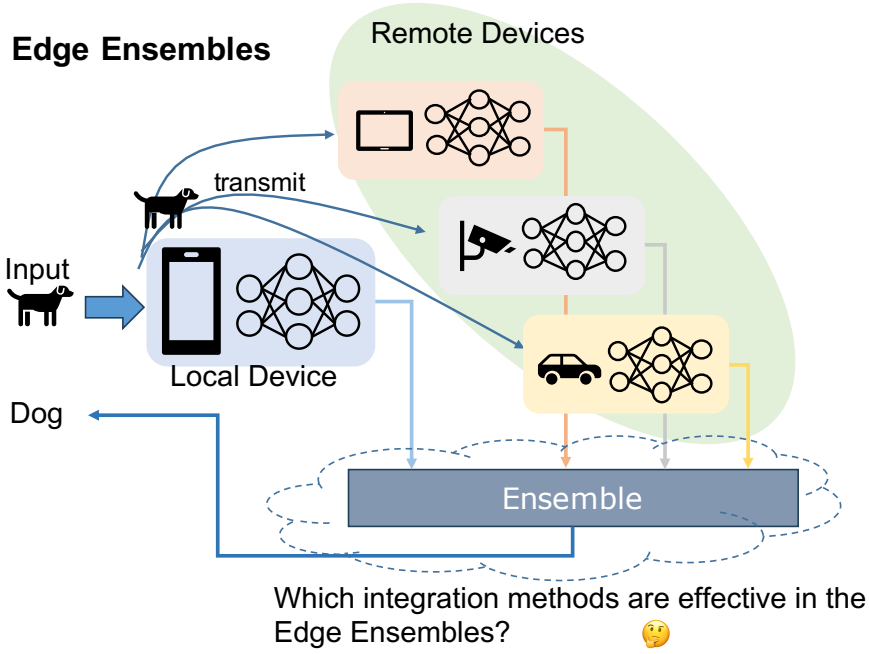

Fig. 4.1: An overview of Edge Ensembles. In this system, the local device collects data and sends it to nearby devices for inference. After inference on each device, the local device gathers and integrates all results, improving accuracy compared to a single prediction. This study examined model integration strategies to improve Edge Ensembles.

on a central server. In Edge Ensembles, communication with other devices incurs additional latency. However, the advantage of ensemble methods, which allow parallel processing of each device, enables significantly lower latency than sequential approaches that involve splitting a large model and running it on multiple devices. Furthermore, Malka et al. [43] proposed quantizing the test data before transmitting it to mitigate network overhead. However, if there are slower devices in the ensemble group, it can significantly reduce the overall throughput. Therefore, Feng et al. [42] proposed network search algorithms and knowledge distillation-based training methods in an Edge Ensemble composed of heterogeneous devices with performance disparities. These approaches aim to ensure that each device meets the constrained inference processing time. In addition, privacy concerns arise in Edge Ensembles due to data transmission to other devices. Yilmaz et al. [74] proposed the security-enhanced method where each device adds noise to the inference results to prevent leakage of information about the model on that device. This approach is similar to FL, where models train collaboratively while keeping their individual information

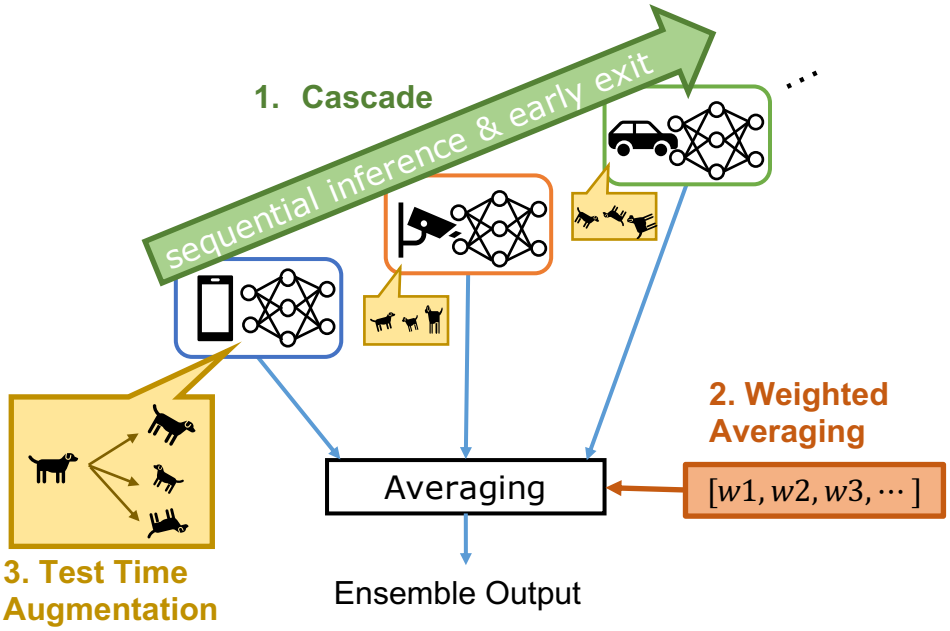

Fig. 4.2: An overview of this study. This study investigates the effectiveness of three existing ensemble methods, namely 1. cascade, 2. weighted averaging, and 3. TTA as improvement techniques for Edge Ensembles. It examines their impact on Edge Ensembles and proposed enhancement methods specifically tailored for Edge Ensembles.

confidential. Similarly, in this method, models collaboratively conduct inference while preserving the confidentiality of their information.

This study focuses on the integration of models for improving Edge Ensembles. In previous Edge Ensembles, inference results from each model are aggregated by majority voting or averaging over noisy outputs [74], averaging the output of each model [6], or weighted averaging based on the F1 score [42] or accuracy [43] as the importance of each model. The following section explores other model integration methods that can further enhance Edge Ensembles.

4.2.2 Ensemble Integration Methods

There are various existing ensemble methods. Traditional ensemble methods include learning strategies of base learners, such as bagging and boosting, as well as integration techniques for combining the outputs of base learners, like majority voting and weighted averaging [29, 75]. There are also methods covering the entire process, from learning to

integrating base learners, such as MoE [53]. In the context of DNNs, implicit ensemble techniques such as dropout [50] and stochastic depth [80] are available. This study focuses on integrating individually trained models using standard training methods without relying on ensemble learning strategies to preserve the advantage of Edge Ensembles, which allows single inference even when communication with another device is lost. This work considers methods of integrating models to improve Edge Ensembles from the following three approaches:

1. Integration method with high efficiency

One of the primary techniques for efficient ensemble integration is ensemble selection. [75, 76, 81]. Ensemble selection methods involve selectively reducing the number of base learners used in inference. However, in Edge Ensembles, ensemble selection methods may not always be feasible or suitable due to the potential unavailability of the desired models or the concentration of access on specific devices with outstanding models. On the other hand, cascade is a method that does not involve base learner selection but dynamically terminates inference based on the confidence of the inference results. As a result, the cascade can be regarded as one of the approaches for dynamic ensemble selection. This study employs cascade as an integration method to enhance efficiency because of its simplicity. It can dynamically reduce the number of models used in inference just by setting the inference terminating threshold in advance.

2. Integration method to improve accuracy

Integrating each output with weighted averaging, also used in previous Edge Ensembles, is generally effective in terms of accuracy. Comparing various integration techniques, such as majority voting and unweighted averaging in ensembling convolutional neural networks (CNNs), the weighted averaging method based on the Super Learner (SL) achieves the highest accuracy [77]. This study examines weighting methods other than those used in previous Edge Ensembles to improve accuracy.

3. Integration of base learners with improved accuracy

In general, improving the accuracy of each base learner leads to an enhancement in the accuracy of ensembles. This study employs TTA as a method to improve the accuracy of a single base learner at inference. TTA is an ensemble method conducted on a single model, which aggregates the inference results for each augmented test data.

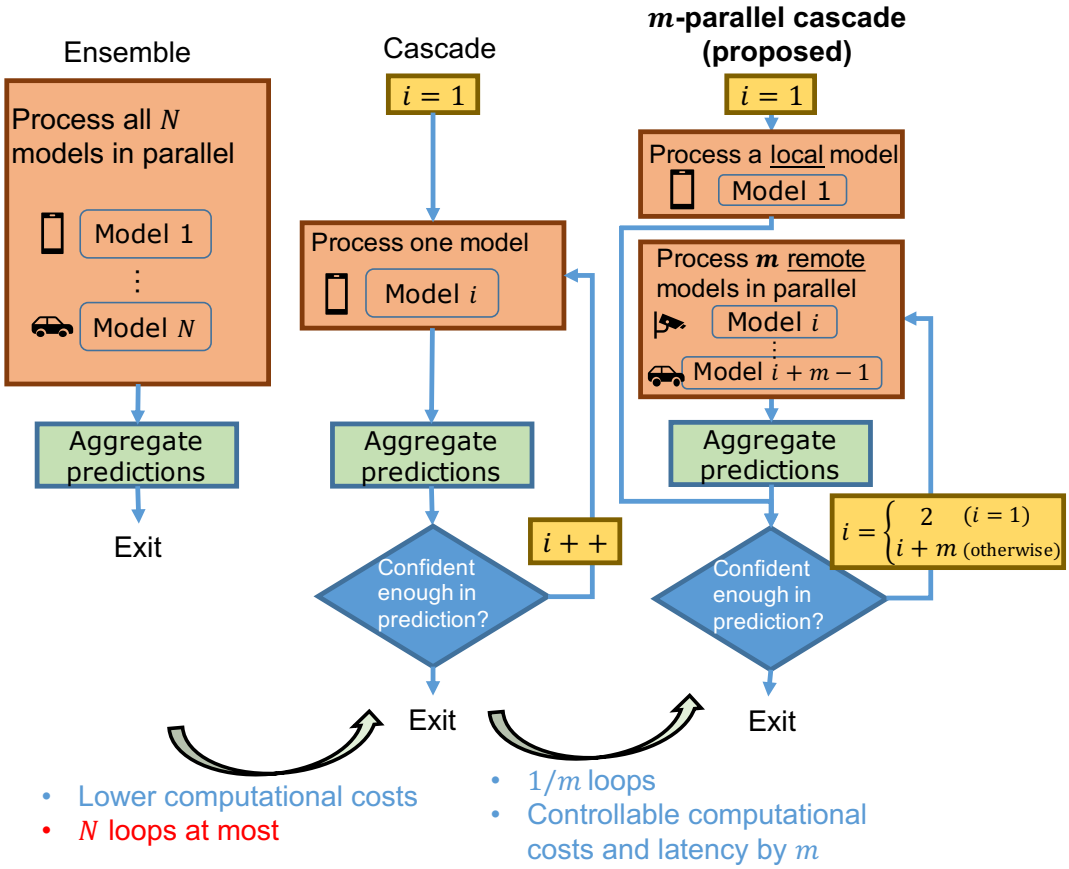

Fig. 4.3: Flows of inference for three methods: simple ensemble (left), cascade (middle), and m-parallel cascade (right, proposed method). The cascade reduces computational costs by conducting sequential inferences and stopping midway, but it cannot parallelize each inference processing, which may lead to increased latency. The proposed method, m-parallel cascade, mitigates latency increase by parallelizing inference for m models.

Fig. 4.2 represents the integration techniques used in this study to improve Edge Ensembles: cascade, weighted averaging, and TTA. The figure indicates the parts of the ensemble where each method is applied. The following sections describe each method.

Cascades

Cascades are techniques that reduce computational complexity by sequentially processing the inferences of base learners and terminating the process at intermediate stages. The original cascade [47] does not aggregate the inference results for each stage, while this

study employs a technique similar to Soft Cascade [48], which propagates information to the next stage, to use in the context of ensemble. Fig. 4.3 compares the inference process by the cascade and simple ensemble. The left side of the figure represents the ensemble, which conducts inferences of all base learners simultaneously and aggregates their results. The center of the figure represents the cascade, which sequentially processes the inference of each base learner and aggregates their results. If the cascade determines that the aggregated result exceeds a certain confidence threshold, it terminates the inference at that point. Compared to the ensemble, the cascade allows for inference with fewer models, especially simpler data. As a result, the average computational complexity decreases.

Cascades are techniques that have been used since before the advancements of DNNs. In the context of DNNs, the cascade has been applied to multiple EfficientNet models in recent years, resulting in a 5.5x speedup for online processing at 5.4x fewer FLOPs while maintaining accuracy [82]. The cascade in Edge Ensembles decreases the number of models required, thereby increasing the number of unused devices. By utilizing these unused devices in another ensemble group, the overall average accuracy of the system can be improved as a result. However, there is a concern about latency degradation when inferring models sequentially. Therefore, in addition to evaluating conventional cascade, this study proposes a method to mitigate cascade latency.

Weighted Averaging

The weighted averaging assigns weights to each base learner when averaging their output class probabilities. While unweighted averaging is a reasonable approach when the performance of each DNN model is roughly the same, it is not optimal when the base learners consist of heterogeneous models [29]. There are several existing weighting methods for base learners including approaches based on Bayesian theory [39–41], the performance of each base learner [42–45], and the utilization of a meta-learner [38, 46].

This study employs two approaches for Edge Ensembles. The first approach utilizes SL, a meta-learning technique, which has demonstrated superior accuracy in CNN ensemble [77]. The second approach exploits the performance of each base learner, inspired by [42], the conventional Edge Ensemble that uses F1 scores as a performance measure. The following paragraphs explain the weighting methods based on SL and F1 scores.

(1) Super Learner-based weighting

SL [46] is a meta-learning technique that learns the weighted combination of each base learner based on their outputs. SL learns the optimal weights for each base learner via

V-fold cross-validation. For the classification in DNN ensembles, the loss function of SL defined in [77] is represented as follows:

$$Loss_{SL}(\vec{w}) = \sum_{v=1}^{V} \sum_{i \in val(v)} l\left(y_i, \sum_{j=1}^{N} w_j \vec{z_{i,j}} \right), \tag{4.2.1}$$

where N is the number of base learners, $\vec{w} = [w_1, w_2, ...w_N]$ is the weight vector, y_i is the class of the i-th data, $val(v)$ is the set of indices of the data in the v-th fold, $\vec{z_{i,j}}$ is the output vector before softmax from the j-th base learner of the i-th data, and l is the classification loss such as cross-entropy loss. SL learns the weight vector that minimizes the loss. In the inference phase, SL gets the class for the i-th data by

$$\text{class}_i = \arg\max_k \sum_{j=1}^{N} w_j z_{i,j,k}. \tag{4.2.2}$$

where $z_{i,j,k}$ is the output value of k-th class from the j-th base learner for the i-th data.

Performing cross-validation multiple times and constructing models at each iteration can be time-consuming. However, it has been shown that SL is effective in finding effective weights even when using a single validation set [77, 83]. This study applied the same method in [77] to model constructions assuming Edge Ensembles.

(2) F1 score-based weighting

Previous Edge Ensembles utilize the F1 score-based weighting method [42]. This method uses confidence scores for each class of each model as the F1 score:

$$c_{j,k} = \frac{2\,\text{Precision}_{j,k}\,\text{Recall}_{j,k}}{\text{Precision}_{j,k} + \text{Recall}_{j,k}}, \tag{4.2.3}$$

where $c_{j,k}$ is the confidence score of the j-th base learner for the k-th class, and $\text{Precision}_{j,k}$ and $\text{Recall}_{j,k}$ are the precision and recall of the k-th class on the j-th base learner for the validation set. Then, the prediction class is calculated by

$$\text{class}_i = \arg\max_k \sum_{j=1}^{N} c_{j,k}\, \sigma\left(\vec{z_{i,j}}\right)_k, \tag{4.2.4}$$

where σ represents softmax function, and thus $\sigma\left(\vec{z_{i,j}}\right)_k$ means the output probability of k-th class from the j-th base learner for the i-th data.

This approach is effective when there is a bias in performance among classes within a model. However, it may cause overfitting if the number of validation sets is insufficient or if

the number of classes is too large. For example, when using the CIFAR-100 dataset, which has 100 classes, and choosing 5,000 images as the validation set, which is half of the test set, there are only 50 images per class. This is not sufficient to determine the performance for each class. Expanding on this approach, this study proposes a weighting method based on the accuracy of whole classes in a model as the general-purpose method not limited to data sets.

Test Time Augmentation

TTA is a technique applying data augmentation to the test data during inference and aggregating inference results for each augmented data to improve accuracy. TTA can be considered as an ensemble of inference data in a single model. Although less significant than the accuracy improvement of TTA on a single model, executing TTA on multiple models and aggregating all their results improves accuracy compared to ensembles without TTA [61]. This research investigate the utility of TTA in Edge Ensembles where the available number of devices varies.

Additionally, there are researches focused on finding effective TTA transformation policies [84–86]. One approach is the Greedy Policy Search (GPS) [84], which employs a greedy search strategy. Other methods involve learning policy weights through gradient descent [85] or using a separate network for policy predictions [86].

This study uses GPS, which is the least computationally expensive, to explore the TTA policy for each model and then investigate whether Edge Ensembles using TTA are effective.

4.3 Proposed Methods

This study examines the efficacy of three existing ensemble methods, namely cascade, TTA, and weighted averaging, for Edge Ensembles. Additionally, it proposes enhancements and extensions to these methods specifically tailored for Edge Ensembles. The proposed methods include m-parallel cascade, offering tunable latency and computational complexity for the cascade. The enhancements also include learning methods using ensemble prediction labels, i.e., labels from final ensemble results, instead of ground truth labels, for TTA and weighted averaging. These methods aim to adjust the integration part in real-world field scenarios where obtaining labeled data is challenging, such as in edge environments. A weighting method based on the accuracy of each base learner is also proposed for weighted averaging. The following sections detail each proposed methods.

4.3.1 m-Parallel Cascade

The cascade reduces the average number of models in inference compared to simple ensembles but requires sequential inference processing. It reduces computational complexity and latency compared to the ensemble in environments that can only process models sequentially, such as a single server. However, in Edge Ensembles, where multiple devices can execute in parallel, the cascade's latency is higher than the ensemble's. The increased latency can be problematic since edge inference systems often require response time.

Therefore, this study proposes m-parallel cascade, which mitigates latency for Edge Ensembles. The right side of Fig. 4.3 shows the flow of the m-parallel cascade. The m-parallel cascade processes multiple m models on remote devices simultaneously, whereas original cascades process one model at a time. This method is just between ensemble and cascade. Note that this method does not parallel the inference in the local device, which is the first iteration of the cascade process. This is because cascades can terminate the inference process with only fast local inference for simple data, eliminating latency in data transfer. Since m-parallel cascade infer m models at once, the average number of models used increases, but the latency decreases compared to the original cascade. By adjusting the parallelism hyperparameter m, it is possible to reduce the number of models in inference while satisfying the required response time. Section 4.4.2 analyzes the accuracy, computational complexity, and latency for varying m.

4.3.2 Ensemble Label-Based Learning

This section introduces ensemble label-based learning for TTA policies and weights in weighted averaging. This approach is specifically designed for Edge Ensembles, collaborative inference systems deployed in edge environments where obtaining labeled data is challenging. Instead of relying on ground truth labels, the method utilizes ensemble prediction labels for adapting the model integration part. The inspiration for this method comes from *Dynamic Deep Ensemble Management* used in [42], which adjusts the participation probability of each device in the inference based on its contribution, as measured by comparing each device's inference result with the final ensemble result. Fig. 4.4 compares the usual method, which uses ground truth labels, and the proposed method, which uses ensemble prediction labels. While the usual learning method is based on the loss computed with the ground truth labels of the training data and the predictions of each base learner, the proposed method uses ensemble prediction labels instead of ground truth labels. The following two paragraphs detail learning TTA policies and weighting for each base learner with ensemble prediction labels.

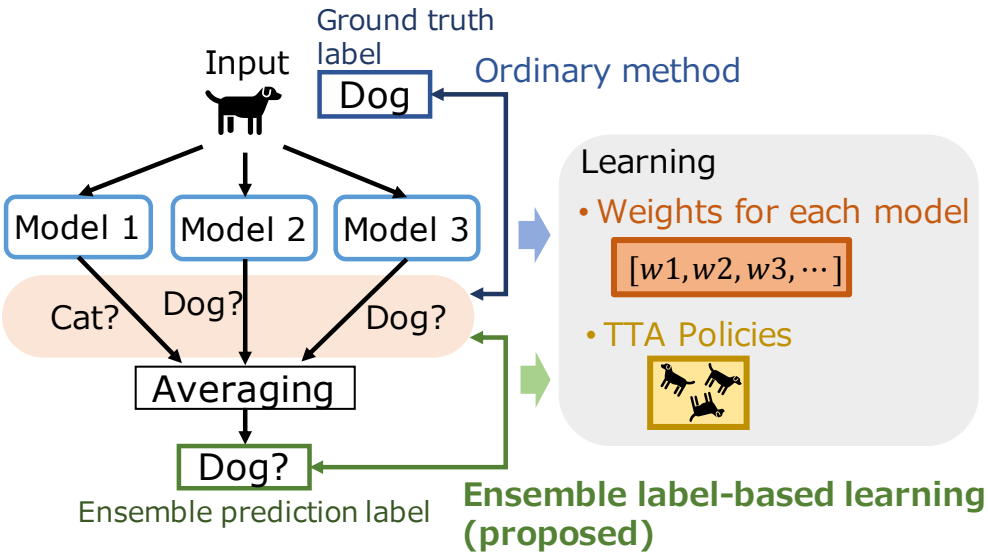

Fig. 4.4: Learning methods for weighted averaging weights and TTA policy. In the conventional approach, each learning process uses ground truth labels, whereas the proposed method utilizes labels from ensemble predictions. This approach addresses the challenges of obtaining ground truth labels in edge environments.

GPS with Ensemble Prediction Labels

This study uses GPS, a greedy algorithm-based lightweight technique, as a policy search method for TTA. The proposed method uses the prediction labels from ensemble results without TTA instead of the ground truth labels as the training data in GPS. Each base learner learns a policy that makes its inference result closer to the ensemble prediction result, which would improve the accuracy of a single model. On the other hand, since this method relies on ensemble results, it is uncertain whether ensembling all results from each base learner using TTA based on this approach would actually lead to an improvement in accuracy. Section 4.4.3 analyzes the effectiveness of this method through experiments.

Weighting Methods with Ensemble Prediction Labels

As in the GPS case above, this method uses ensemble prediction labels instead of ground truth labels for learning weights in weighted averaging. Weighting methods based on each base learner's performance, such as F1 score or accuracy, can dynamically update the weights during inference by comparing each base learner's prediction with ensemble

prediction labels. However, it is uncertain whether learning weights using ensemble prediction labels leads to effective weighting. Therefore, this study experimentally validate the effectiveness of this approach.

4.3.3 Accuracy-Based Weighting Method

This study employs two types of weighting methods, one based on SL and the other based on the performance of each base learner. The former method follows the same approach as [77]. In the latter method, previous research [42] had employed class-specific F1-scores for the performance metric as indicated in Eq. 4.2.4. However, there is a concern that this approach may not function effectively when there is insufficient data to calculate F1 scores for each class. Fig. 4.9 compares the accuracy of the CIFAR-100 dataset between scenarios with no weighting, weighting based on class-specific F1 scores, and weighting based on the macro average of F1 scores across all classes. When using class-specific F1 score-based weights, the accuracy decreased compared to not using weights. However, when using the macro-averaged F1 score for weighting, accuracy was slightly improved compared to not using weights. These results suggest that overfitting of the weights occurred when calculating class-specific F1 scores.

Therefore, this study uses accuracy-based (ACC-based) weighting, which simplifies the implementation without significantly altering the functionality compared to using average class-specific F1 scores for weighting. Furthermore, this methods does not use accuracy as weights directly to improve performance but rather employ a function to get the appropriate weights. It determines the weight w_i for the i-th base learner as an ACC-based method using softmax with temperature as follows:

$$w_i = \frac{\exp\left(\frac{acc_i}{T}\right)}{\sum_{j=1}^{N} \exp\left(\frac{acc_j}{T}\right)}, \tag{4.3.1}$$

where acc_i is the accuracy of the i-th base learner, and T is the temperature in softmax. To obtain the optimal weights, adjusting the softmax temperature T is necessary. Section 4.4.2 examines the accuracy under varying T.

4.4 Experiments

This section investigates the performance improvement effects of existing model integration methods in Edge Ensemble and evaluates the effectiveness of the proposed method through experiments. First, it outlines the system design and evaluation metrics of Edge Ensembles.

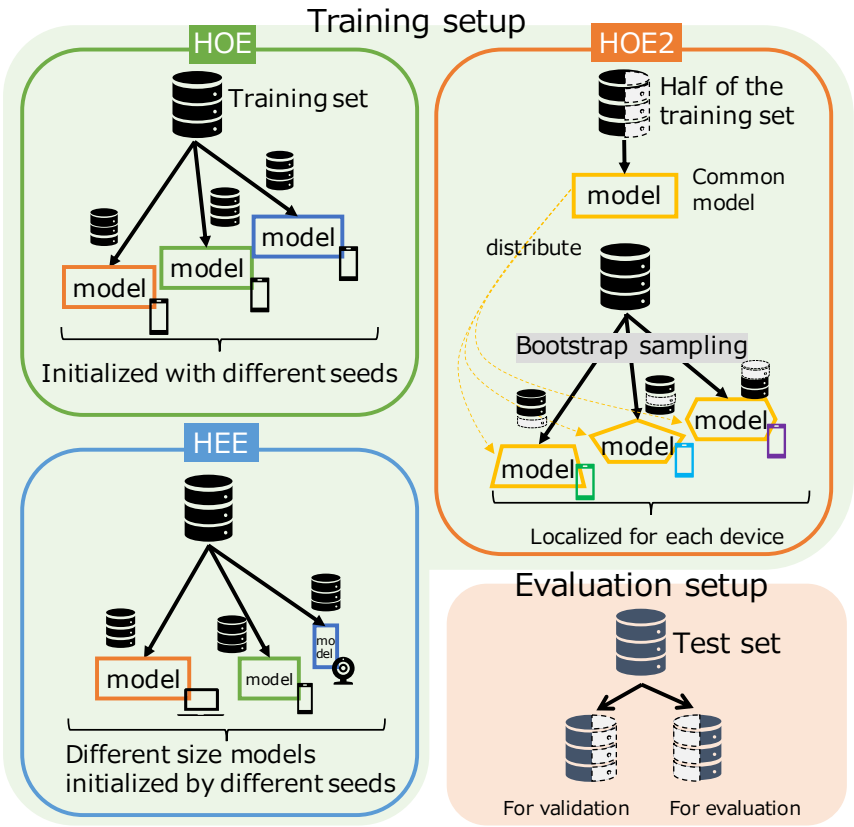

Fig. 4.5: Schematic diagram of the experimental setup. HOE assumes each device has its own separately generated models, and HOE2 assumes a common model is localized on each device. HEE takes into account the varying model size requirements of each device. In the evaluation, this study divides the test set into two halves, one for validation and the other for evaluation.

Next, it explores the appropriate hyperparameters for the proposed method. Following that, it evaluates the performance of each method and concludes with a discussion.

4.4.1 System Design and Evaluation Setup

This study focuses on how to combine models in Edge Ensembles rather than training individual models. Therefore, the evaluation process prepares the sets of models assuming Edge Ensembles in advance and evaluates the inference performance for each integration methods. This section first present the model configurations designed for Edge Ensembles and the datasets used for evaluation, then describes the settings of the ensemble methods

used in this study.

Models and Datasets

This study consider three model configurations of Edge Ensembles: 1. Homogeneous Ensemble (HOE), 2. HOE2, and 3. Heterogeneous Ensemble (HEE). Fig. 4.5 shows the schematic of the training strategies for each model configuration. The HOE setting consists of 16 ResNet18 models, each of which trains with the entire training data from different initial weights generated by each seed. The HOE2 setting also consists of 16 ResNet18 models. However, each model is based on a single ResNet18 trained for half of the total epochs using half of the training data. Then, each model trains for the remaining half of the total epochs using bootstrap-sampled data from a whole training set. This configuration assumes a scenario where a common model trained on the server is localized on the edge. The HEE setting consists of lightweight variants of ResNet18 models based on five different configurations. This study reduce the channel sizes of each layer in ResNet18 by multiplying a parameter x, which is a reduction factor relative to the original channel size. In the HEE setting, it set x to 0.2, 0.4, 0.6, 0.8, and 1.0 (original ResNet18) and create 2, 3, 6, 3, and 2 models for each x, resulting in 16 models. Each model trains using the entire training data from different initial weights. This model configuration accounts for the scenario where the Edge Ensemble comprises edge devices with varying model size requirements.

For evaluation, this study utilize CINIC-10, CIFAR-100, and Tiny ImageNet datasets. CINIC-10 is a 10-class dataset that combines samples from CIFAR-10 and ImageNet. Tiny ImageNet is a 200-class dataset that contains extracted data from ImageNet. For all datasets, each model is trained for 100 epochs. The optimizer is SGD with a momentum of 0.9. A weight decay of 5×10^{-4} is applied. The learning rate is set to 0.1 with a cosine annealing schedule. The evaluation process trains each model using all the training sets and divides the test sets in half, using one half for evaluation and the other half as a validation set. The validation set is used to adjust various hyperparameters and learn TTA policies and weights for weighted averaging.

Setup of Ensemble Methods

This work uses the simple ensemble with unweighted averaging for each base learner's output probability as the baseline for evaluation, and this dissertation conveniently calls this method the vanilla ensemble. Following paragraphs describe the implementation and parameter settings in ensembles with the cascade, TTA, and weighted averaging used in

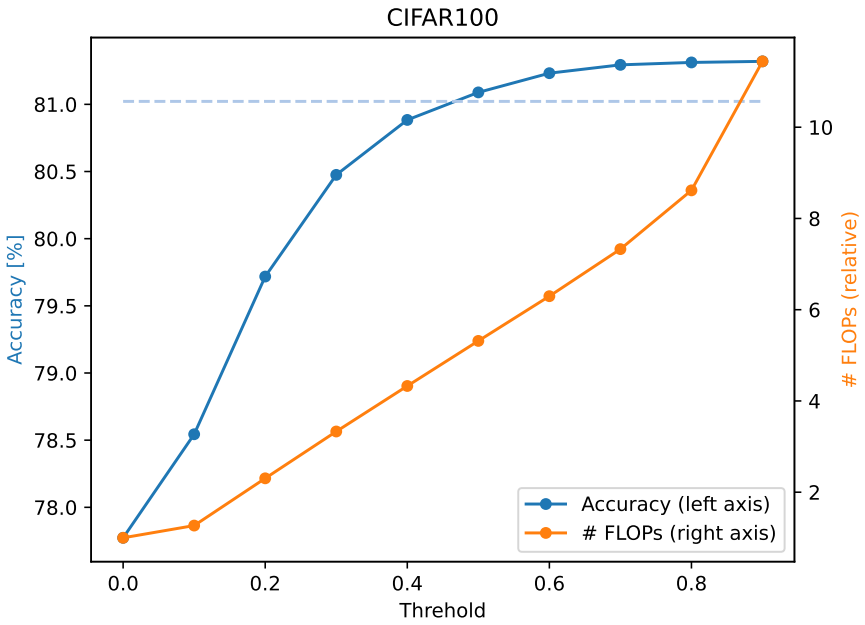

Fig. 4.6: Relationship between cascade termination threshold, accuracy, and computational costs. The left axis and the blue line represent accuracy, while the right axis and the orange line represent relative computational costs. This work selects a threshold that reduces the computational costs the most within a 0.3% accuracy reduction (indicated by the dotted line) from the maximum accuracy.

this study.

For the cascade, it is necessary to set up a confidence measure function and an inference termination threshold. This work uses the confidence measure based on the maximum probability in the prediction, whose effectiveness has been demonstrated in [61]. Since it is not always possible to infer a specific device order in Edge Ensembles, this work sets the threshold to be constant within a series of the cascade. Fig. 4.6 shows the accuracy and computational complexity of the HOE for the CIFAR-100 dataset when varying the threshold in cascade. Lowering the threshold leads to a decrease in accuracy but reduces computational complexity. The appropriate threshold depends on the required accuracy of the target task. This evaluation defines a tolerance range for accuracy degradation within a range of 0.3%, which is sufficiently smaller than the accuracy improvement effect achieved by the ensemble. The evaluation process minimizes computational costs by selecting the cascade threshold to keep the accuracy decrease on the validation set within 0.3% of the

vanilla ensemble.

For TTA, this study uses GPS for each model to find valid policies from the candidate policy set before inference. This study utilizes a candidate policy set consisting of 276 different transformations. This set includes the 23 transformations proposed by RandAugment [87] with a transformation magnitude of 9, which consider the sign of the magnitude, as well as an additional transformation of horizontal flip. By combining two transformations from these 24 options, it generates ${}_{24}C_2 = 276$ unique transformations. Note that this approach do not consider the order of the combinations to prevent an excessive number of candidate policies. In the inference phase, it applies the top 10 transformation policies found by GPS for each model.

In weighted averaging, this study uses SL-based and ACC-based weighting methods. SL-based weighting uses a 1x1 convolutional layer, similar to the approach in [77], to assign a single weight to each model. The SL trains through back-propagation with the validation set. ACC-based weighting divides the validation set in half, calculating the accuracies of each base learner with one half and determining the optimal T with the other half. After determining T, the method assigns weights based on the accuracy of the entire validation set.

In order to avoid the complexity of the search space, each method has only one change from the vanilla ensemble. For example, in the case of the cascade, only the part that integrates base learners sequentially is changed from the vanilla ensemble, and the part that uses unweighted averaging to aggregate inference results is the same.

Setup of Evaluation Metrics

As evaluation metrics, this study assess the accuracy of all the methods through experiments. Furthermore, it estimate how computational costs and latency change compared to the original Edge Ensemble.

First, regarding computational costs, weighted averaging involves multiplying each model's predictions by their respective weights, so there is little change compared to the original Edge Ensemble. In the case of TTA, the computational costs increase by the number of TTA transformation policies applied. In our experiment, it applies 10 transformation policies, resulting in a 10-fold increase in computational costs. As for the cascade, the computational costs vary for each data point, depending on the number of models used for inference. This analysis estimate the average change in the number of FLOPs (Floating-Point Operations) for cascade compared to the original Edge Ensemble.

Next, regarding latency, for weighted averaging, as with the computational cost, it is almost equivalent to the original Edge Ensemble. For TTA and cascade, latency changes

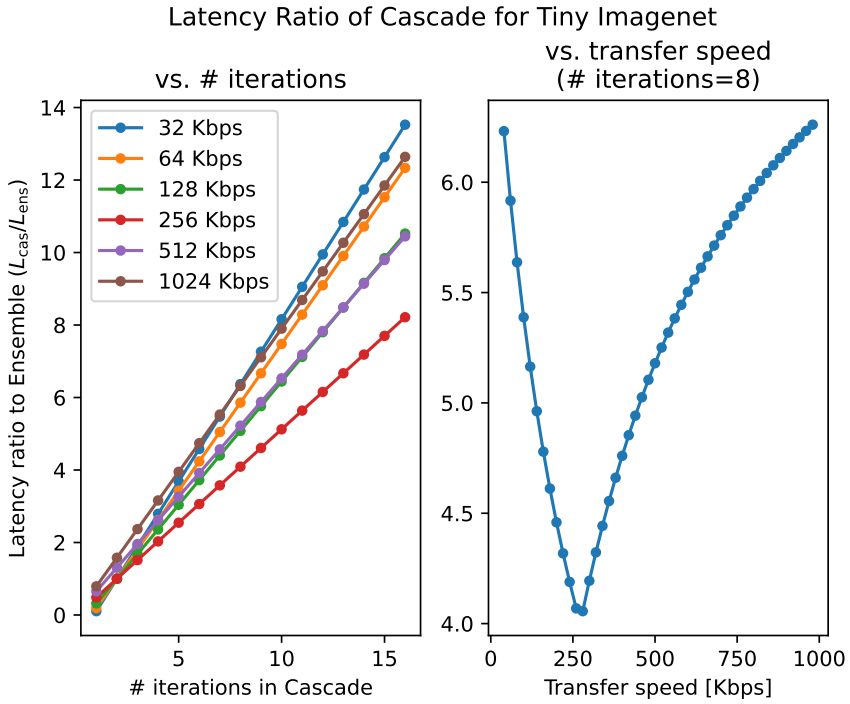

Fig. 4.7: Latency ratio of cascade compared to a regular edge ensemble for various transfer speeds and cascade iteration counts. When data transfer time and inference time are equal, the parallel execution of these processes is most effective, resulting in the minimal latency ratio. When the number of iterations in the cascade is 1, the cascade ends with local device inference only. Therefore, the latency is lower than that of the original Edge Ensemble.

compared to the original Edge Ensemble. To estimate these changes, the process first briefly estimate the latency in the original Edge Ensemble. A detailed latency analysis is conducted in [43], but here, to provide a more accessible estimate of how the method employed in this study impacts the Edge Ensemble's latency, this approach utilizes a simplified latency model. Assuming that data transfer times and inference times are consistent across all devices, the latency of the Edge Ensemble L_{ens} is represented as follows:

$$L_{\text{ens}} = T_{\text{data}} + \tau, \tag{4.4.1}$$

where T_{data} represents the transfer time of the inference data to other devices, and τ represents the inference time of each device. This latency equation represents the time when a local device sends data to other remote devices simultaneously, and remote devices perform inference simultaneously. This work ignore the time required to return the inference

results in the equation because the data size of the inference results is considerably smaller than the inference data. In the case of TTA, the τ increases by the number of TTA transformation policies applied. Therefore, it can be expressed by the following equation:

$$L_{\mathrm{TTA}} = T_{\mathrm{data}} + k\tau, \tag{4.4.2}$$

where k represents the number of TTA policies applied. The increase in L_{TTA} compared to L_{ens} depends on the ratio of τ to T_{data}, but once the values are determined, it can be statically calculated. If τ is dominant, it increases by up to a maximum of k times. In the case of cascade, the latency varies depending on how data transfer and inference processing are controlled sequentially. Here, this analysis assumes that the inference of data on a certain device and the data transmission to the next inference device are carried out in parallel simultaneously. Then, the latency is represented by the following equation:

$$L_{\mathrm{cas}} = \begin{cases} I\tau + (T_{\mathrm{data}} - \tau)(I-1) & (T_{\mathrm{data}} - \tau \geq 0) \\ I\tau & (T_{\mathrm{data}} - \tau < 0) \end{cases}, \tag{4.4.3}$$

where I represents the number of iterations in the cascade. By performing inference and data transmission simultaneously, it is possible to hide the time taken for data transmission. Fig. 4.7 summarizes latency changes from the original Edge Ensemble for various transfer speeds using Tiny ImageNet dataset. This analysis estimates τ using a typical model inference speed on edge devices of 700 milliseconds for an image sized 224x224, based on measurements in [88], and adjusts this time for the Tiny ImageNet size of 64x64. Furthermore, the analysis considers transfer speeds range from tens to hundreds of kbps, based on common communication methods used in edge environments, as used in [89]. The calculation of data size uses the JPEG images in the Tiny ImageNet dataset. The latency of the cascade varies depending on the number of cascade iterations. Therefore, the evaluation measures the number of cascade iterations during experiments and calculates the latency by averaging the latency ratios for the six data transfer speeds shown in the left figure of Fig. 4.7.

4.4.2 Hyperparameter Analysis

This section analyzes two hyperparameters of the proposed method, the parallelism m in the m-parallel cascade and the temperature T in the ACC-based weighting method.

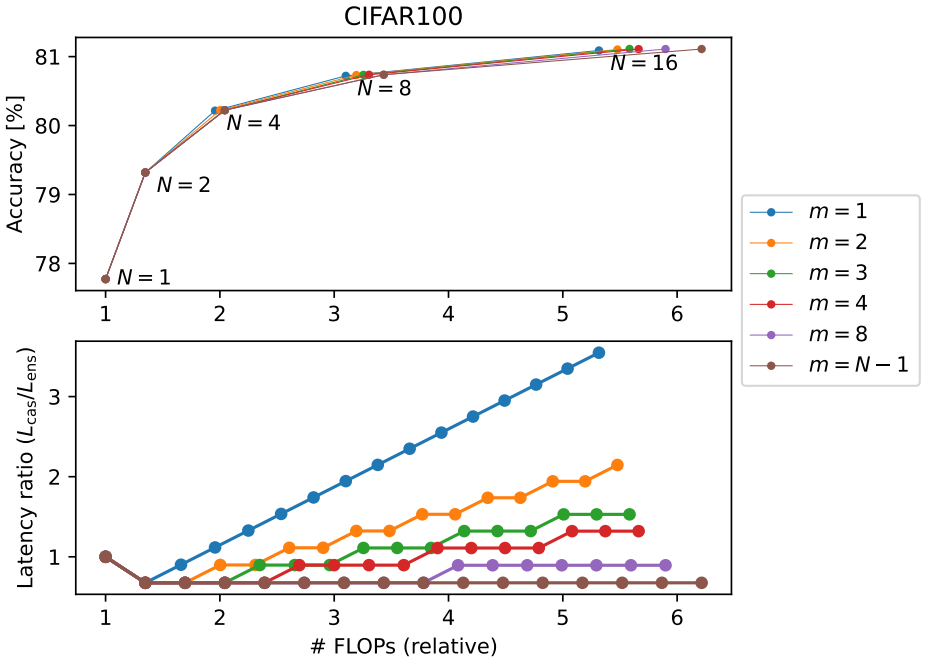

Fig. 4.8: Latency and accuracy of m-parallel cascade for varying m in the case of the HOE setting with the CIFAR-100 dataset. N represents the total number of models in the cascade. Each dot in the bottom figure corresponds to an increasing N from 1 to 16. When the number of models is 1, both cascade and ensemble have the same latency because they use only a local model for inference. Depending on the number of models, cascade has an advantage over the ensemble regarding latency because cascade can terminate inference with only a local model for simple data.

Parallelism in m-Parallel Cascade

Fig. 4.8 shows the accuracy and latency of the proposed method for m-parallel cascade for varying the parallelism parameter m. The top part of the figure shows the accuracy, and the bottom part shows the latency. The horizontal axis denotes the average number of models participating in inference, expressed as relative FLOPs. Because similar trends appeared regardless of the dataset and model configuration, the figure only shows the data for the CIFAR-100 in the HOE setting. The analysis shows that ensemble accuracy for the same number of models remains almost the same as the change in m while the computational complexity increases and latency decreases. The results also indicate that the latency is

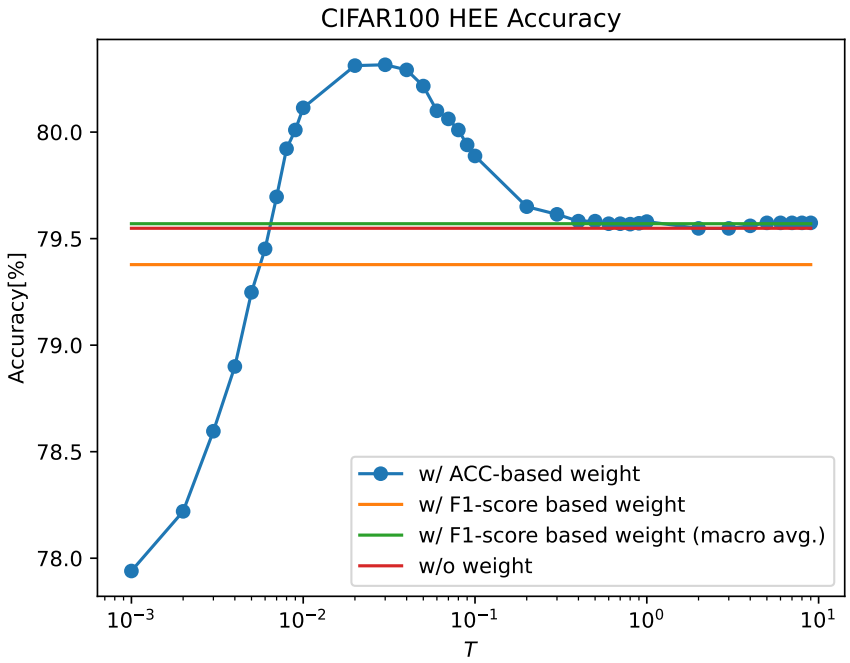

Fig. 4.9: Accuracy using ACC-based weighting for varying T in Equation 4.3.1 in the case of the HEE model configuration with the CIFAR-100 dataset. The blue line shows the accuracy of the ACC-based weighted ensemble with different values of T, and the red line represents the accuracy of the vanilla ensemble as a baseline. The orange line represents the accuracy with the class-specific F1 score-based weighting method, while the green line represents the accuracy with weighting based on the averaged F1 score across all classes.

most efficient when m is a divisor of the number of remote devices used for inference. Section 4.4.3 evaluates the trade-off between computational complexity and latency for cases of $m = 1, 2, 4$, and 8 for the cascade with 16 models.

Temperature in ACC-Based Weighting Methods

Fig. 4.9 shows the accuracy for varying temperature T from 0.001 to 9 for the ACC-based weighting function in Equation 4.3.1. Similar trends appeared across all datasets. However, for model configurations, the HOE and HOE2 settings had little effect on ACC-based weight assignment (details will be discussed in Section 4.4.3). Therefore, the figure focuses on HEE for CIFAR-100. For HEE on all datasets, too low values for T significantly degraded

TABLE 4.1: Accuracy of the single model and the vanilla ensemble with 16 models for the baseline.

Dataset	Accuracy[%] (diff with single)					
	HOE		HOE2		HEE	
	single	Ensemble	single	Ensemble	single	Ensemble
CINIC-10	87.1	89.5 (+2.4)	84.4	87.0 (+2.6)	85.3	88.7 (+3.4)
CIFAR-100	77.3	81.0 (+3.7)	72.4	76.0 (+3.6)	73.6	79.6 (+6.0)
Tiny ImageNet	65.1	72.2 (+7.1)	60.2	66.7 (+6.5)	60.9	69.4 (+8.5)

TABLE 4.2: Accuracy of each model composing the HEE setting.

Dataset	Accuracy[%]				
	$x = 0.2$	$x = 0.4$	$x = 0.6$	$x = 0.8$	$x = 1$
CINIC-10	81.0	84.5	85.9	86.5	87.2
CIFAR-100	66.9	71.1	74.6	76.1	77.4
Tiny ImageNet	56.3	59.1	60.8	63.4	65.1

accuracy, and too high values for T had no effect, resulting in a T peak between 0.01 and 0.1. Based on the results, the evaluation in Section 4.4.3 uses the value of T that yielded the highest accuracy on the validation set for each dataset and model configuration.

4.4.3 Evaluation

This section conducts experiments to evaluate the effectiveness of the Edge Ensembles when using cascade, weighted averaging, and TTA for model integration. It also experimentally validates the effectiveness of the proposed methods of m-parallel cascade, ensemble label-based learning, and the ACC-based weighting method. Table 4.1 presents the accuracy of the single model and vanilla ensemble consisting of 16 models as baselines for each model configuration and dataset. The column of the single model represents the average accuracy of all the models composing its model configurations. Table 4.2 offers the accuracy of the single model composing the HEE setting for each x.

TABLE 4.3: Accuracy of cascade with 16 models.

Dataset	Accuracy [%] (diff from vanilla ensemble)			
	HOE	HOE2	HEE	
			asc	desc
CINIC-10	89.4 (−0.1)	86.9 (−0.1)	88.6 (−0.1)	88.6 (−0.1)
CIFAR-100	80.7 (−0.3)	75.9 (−0.1)	79.2 (−0.4)	79.3 (−0.3)
Tiny ImageNet	72.0 (−0.2)	66.4 (−0.3)	69.3 (−0.1)	69.4 (−0.0)

TABLE 4.4: FLOPs and latency of cascade with 16 models.

Dataset	FLOPs ratio to vanilla ensemble				Latency ratio to vanilla ensemble			
	HOE	HOE2	HEE		HOE	HOE2	HEE	
			asc	desc			asc	desc
CINIC-10	0.31×	0.34×	0.50×	0.26×	3.24×	3.69×	6.02×	1.77×
CIFAR-100	0.34×	0.41×	0.54×	0.35×	3.60×	4.48×	6.42×	2.85×
Tiny ImageNet	0.58×	0.56×	0.69×	0.63×	6.38×	6.23×	8.01×	6.38×

Cascade

Table 4.3 and Table 4.4 show the accuracy, FLOPs, and latency of the ensemble with cascade for each dataset and model configuration using 16 models. In the tables, values in parentheses for accuracy represent the difference compared to those of the vanilla ensemble. The FLOPs and latency values indicate the ratio when compared to the values of the vanilla ensemble. The analysis determines FLOPs and latency for the HOE and HOE2 settings based on the average number of models participating in inference. For the calculation of FLOPs and latency for the HEE setting, the cascade uses ascending and descending order based on model size criteria due to its composition of models with different structures. In addition, the analysis calculates the FLOPs of each model for each x. However, latency remains constant regardless of x for simplicity because the latency of data communication is independent of x, and the inference time depends on the computational power of each device in addition to x.

The accuracy in all model configurations and datasets remained nearly unchanged while reducing the FLOPs. However, the latency increased. When comparing each model

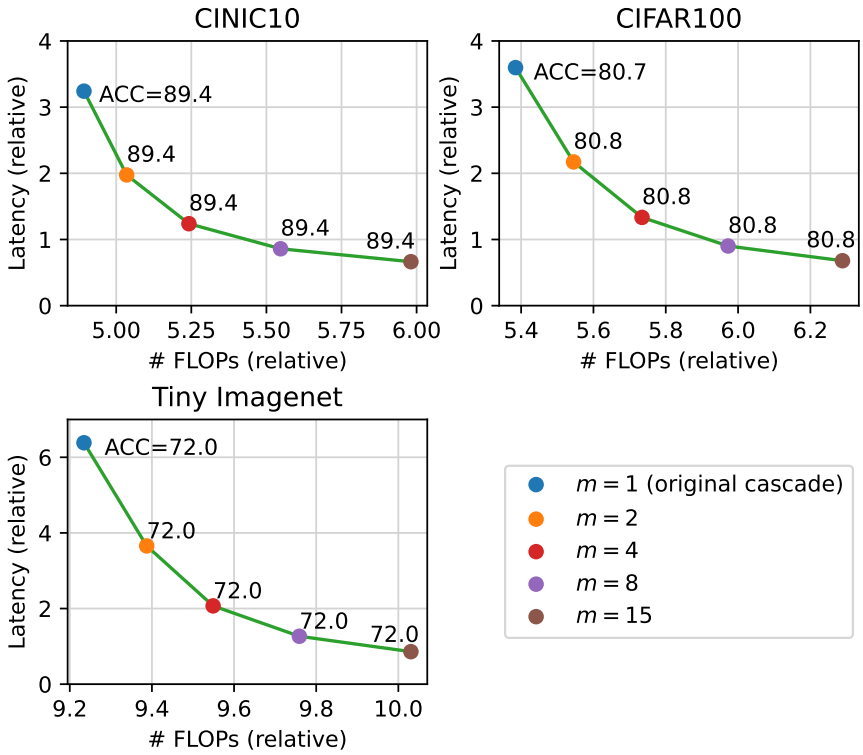

Fig. 4.10: Computational cost and latency of the m-parallel cascade in the HOE setting for each dataset. The vertical axis, representing latency, shows the latency ratio compared to a regular ensemble. The horizontal axis, representing FLOPs, indicates the relative computational cost of the cascade compared to a single model, where the computational cost of a single model is normalized to 1. These values were measured based on the average number of models used in the cascade. While there are minor variations in accuracy based on the value of m, the method preserves overall accuracy and allows for tuning latency and FLOPs.

configuration, we observes that the HOE and HOE2 settings yielded similar results. In the case of HEE, the descending order cascade outperformed the ascending order cascade. This is due to the thresholds used to minimize accuracy drops in the cascade process. In the ascending order cascade, to maintain accuracy, the inference must continue until the later strong models if the earlier weak models make incorrect class predictions. Therefore, a higher termination threshold is necessary, and terminating the inference process becomes more challenging, even for simpler data.

TABLE 4.5: Accuracy of ensemble with weighted averaging.

Dataset	Weighted Averaging Accuracy [%] (diff from the baseline)					
	HOE		HOE2		HEE	
	ACC-based	SL-based	ACC-based	SL-based	ACC-based	SL-based
CINIC-10	89.5 (+0.0)	89.5 (−0.0)	87.0 (+0.0)	87.0 (+0.0)	89.0 (+0.3)	88.9 (+0.2)
CIFAR-100	81.1 (+0.1)	81.1 (+0.1)	76.1 (+0.1)	75.9 (−0.1)	80.3 (+0.7)	79.9 (+0.3)
Tiny ImageNet	72.1 (−0.1)	72.0 (−0.2)	66.7 (+0.0)	66.6 (−0.1)	70.2 (+0.8)	70.5 (+1.1)

Furthermore, this section applies m-parallel cascade to the HOE setting and examines the changes in latency and computational complexity. Fig. 4.10 shows the variation in latency and FLOPs of the m-parallel cascade for each dataset. By varying the value of m, the m-parallel cascade achieved a trade-off between latency and computational complexity while maintaining nearly the same level of accuracy. For instance, when m was 4, the latency was about 2.8 times lower than the original cascade while increasing FLOPs by 1.06 times on average accross all datasets. Compared to the vanilla ensemble, the average amount of computation was 0.43 times, while the increase in latency was controlled to just 1.55 times. Furthermore, when m is greater than or equal to 8, it reduces latency and computation costs while maintaining accuracy compared to the vanilla ensemble. In practical applications, selecting an appropriate value of m based on the acceptable latency, it is possible to reduce computational complexity while maintaining accuracy.

Weighted Averaging

Table 4.5 shows the accuracy in the ensemble with 16 models when using the SL-based and ACC-based weighted averaging for each dataset and model configuration. In the HOE and HOE2 settings, both ACC-based and SL-based weighted averaging did not yield significant improvements. However, in the HEE setting, both methods proved to be effective. This can be attributed to the minimal variations between the performances of models within the HOE and HOE2 settings, which renders the weighting meaningless.

In the HEE setting, when comparing ACC-based and SL-based weighting methods, ACC-based outperformed SL-based in CIFAR-100, while SL-based was superior in Tiny ImageNet. CIFAR-100 is a simpler dataset than Tiny ImageNet. It is considered that ACC-based weighting, which simply prioritizes each prediction based on accuracy, was more effective than performing complex weight learning using SL for simpler data. Furthermore, in the case of CINIC-10, the weighting methods did not yield significant

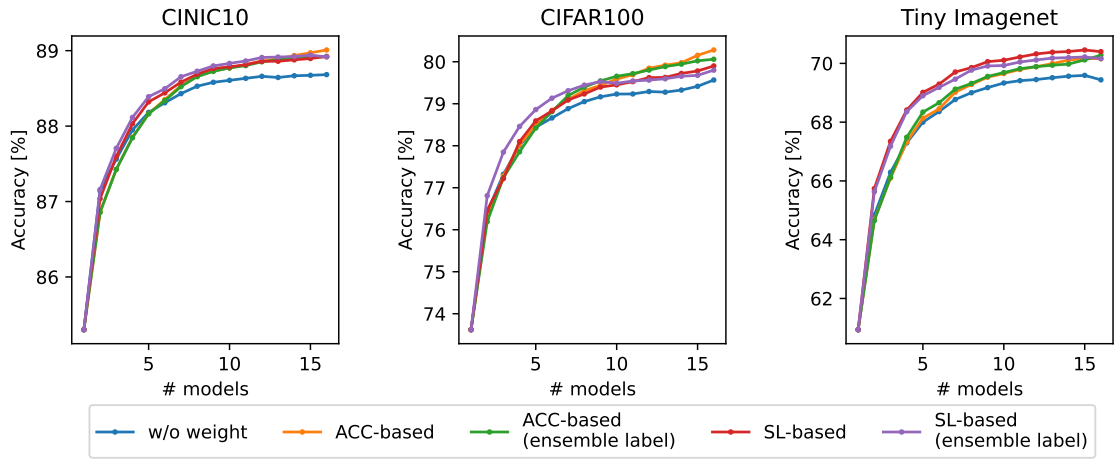

Fig. 4.11: Comparison of ensemble accuracies with and without weighted averaging in the HEE setting for varying numbers of base learners. The blue line represents the vanilla ensemble, while the other colored lines depict ensembles with weighted averaging using each weighting method. For the green and purple lines, ensemble prediction labels are used instead of ground truth labels for weighting. The evaluation process randomly sampled models to measure accuracy after training the weights using all 16 models. While there was some variation in effectiveness among the weighting methods, weighted averaging showed efficacy even with varying the number of base learners. The method using ensemble prediction labels demonstrated nearly equivalent performance to using ground truth labels.

effects. This outcome can be attributed to the fact that CINIC-10 is the easiest dataset and the minimal performance differences between base learners within the HEE setting, as shown in Table 4.1. In the case of HEE, where there were significant effects, the analysis focuses 1) the accuracy changes for the number of base learners and 2) the learning of weights using the proposed training method with ensemble prediction labels.

1. Effectiveness for varying number of base learners

In Edge Ensemble scenarios, not all models on each device are always available due to communication constraints. Therefore, it is essential to verify the effectiveness of weights learned using all models in training when only a subset of models is available in inference. Fig. 4.11 shows the average accuracy in the HEE setting for the change in the number of base learners for the ensemble with weighted averaging. The training phase involves learning weights from all 16 models, while the inference phase selects subsets of models

TABLE 4.6: Accuracy of ensemble with TTA.

Dataset	TTA Accuracy [%] (diff from the baseline)					
	HOE		HOE2		HEE	
	single	Ensemble	single	Ensemble	single	Ensemble
CINIC-10	87.9 (+0.8)	89.5 (+0.0)	85.2 (+0.8)	87.1 (+0.1)	86.4 (+1.1)	88.7 (+0.0)
CIFAR-100	78.0 (+0.7)	81.1 (+0.1)	73.1 (+0.7)	76.2 (+0.2)	75.1 (+1.5)	79.6 (+0.0)
Tiny ImageNet	66.0 (+0.9)	72.2 (+0.0)	61.1 (+0.9)	67.2 (+0.5)	62.6 (+1.7)	69.9 (+0.5)

randomly. The results shows that conducting weighted inference using randomly selecting a subset of models is as effective as using all 16 models. This observation validates the effectiveness of the weighting approach for Edge Ensemble scenarios.

2. Learning with ensemble prediction labels

Fig. 4.11, compares the accuracy of learning with ground truth labels and ensemble prediction labels for ACC-based and SL-based weights. The weighting method of using ensemble prediction labels yielded results that were almost equivalent to using ground truth labels. This experimental confirmation demonstrates the effectiveness of ensemble label-based weighting in situations where weighted averaging is beneficial.

TTA

Table 4.6 shows the accuracy of a single model and an ensemble with 16 models when applying TTA for each dataset and model configuration. Applying TTA to single models improved accuracy consistently across all model configurations. However, in most cases, ensembling did not show significant improvements. Although there was some improvement in ensembling for the HOE2 and HEE settings on Tiny ImageNet, it was still lower than the accuracy improvement achieved by TTA on a single model. These results suggest that ensembling and TTA share some redundant functionalities.

Fig. 4.12 shows the effectiveness of TTA for each total number of base learners in the ensemble for the HOE setting, which had the lowest efficacy of TTA in the ensemble among all model configurations. TTA demonstrated a notable impact when the ensemble had a small number of models, but the effect diminished as the number of models increased. Similar trends were observed for other model configurations and datasets not shown in the figure. These results indicate that TTA can be an effective method to improve accuracy in Edge Ensembles further when the number of available models is limited.

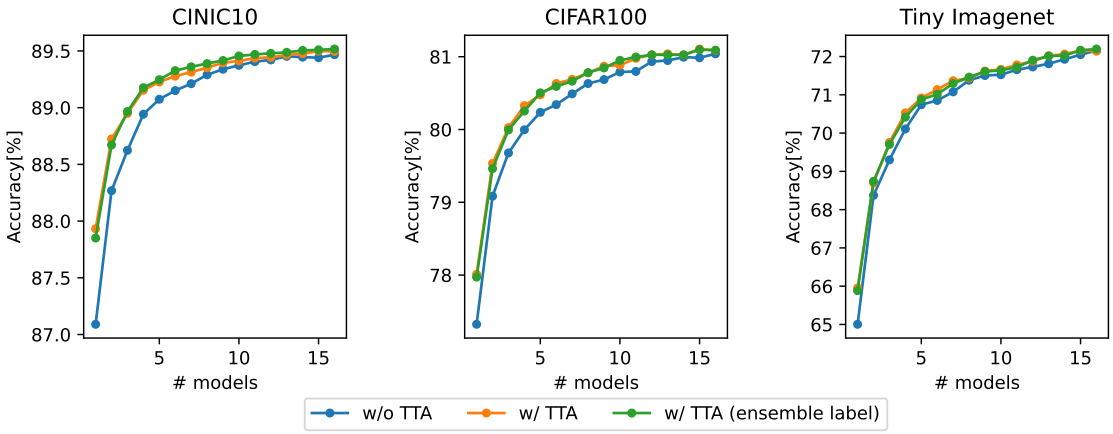

Fig. 4.12: Comparison of ensemble accuracies with and without TTA in the HOE setting for varying numbers of base learners. The blue line represents the vanilla ensemble, the orange line represents the ensemble with TTA, and the green line represents the ensemble with TTA using policies learned from ensemble prediction labels instead of ground truth labels. As the number of base learners increased, the difference in accuracy between using TTA and not using TTA decreased. Furthermore, the effect of TTA was almost equivalent regardless of the difference in the labels used during policy learning.

Fig. 4.12 also shows the accuracy of the proposed ensemble label-based learning, which learn the policy for TTA by GPS using ensemble prediction labels instead of ground truth labels. Although the accuracy is slightly lower for the proposed method compared to the case using ground truth labels, the overall effectiveness of TTA is still evident. This indicates that even in Edge Ensemble scenarios where ground truth labels are unavailable, treating ensemble prediction labels as pseudo-ground truth allows for effective TTA policy learning.

Evaluation of Combinations

This section evaluates the performance when combining these methods. Cascade, TTA, and weighted averaging are three independent methods that can be used simultaneously, but each has distinct characteristics. Cascade is a method that reduces accuracy but decreases computational complexity, while TTA is a technique that increases accuracy at the expense of computational complexity. These two methods have contrasting features, and there are no advantages to using them simultaneously. Therefore, this work does not consider scenarios

TABLE 4.7: Accuracy, FLOPs ratio and Latency ratio of the combination of cascade and weighted averaging.

Dataset	HEE w/ cascade & weighted averaging		
	Accuracy [%] (diff from cascade/weighting/vanilla)	FLOPs ratio (cascade alone)	Latency ratio (cascade alone)
CINIC-10	88.7 (+0.1/−0.3/−0.0)	0.26× (0.26×)	1.84× (1.77×)
CIFAR-100	79.8 (+0.5/−0.5/+0.2)	0.35× (0.35×)	2.93× (2.85×)
Tiny ImageNet	70.4 (+1.0/−0.1/+1.0)	0.60× (0.63×)	5.95× (6.38×)

where they are used together. Consequently, this section evaluates two combinations: the combination of cascade and weighted averaging, and the combination of TTA and weighted averaging. In evaluating each combination, this analysis assesses the HEE setting in which weighted averaging was the most effective among the three model configurations.

1. Combination of cascade and weighted averaging

For the combination of cascade and weighted averaging, this analysis uses cascade with descending order and employed weighting methods that yielded higher accuracy for each dataset. The evaluation relies on the same metrics as those used when applying cascade alone. Table 4.7 presents accuracy, FLOPs ratio, and latency ratio of HEE settings with cascade and weighted averaging. By combining cascade and weighted averaging, FLOPs and latency remained nearly the same level as in the case of cascade alone, while the accuracy matched or exceeded that of the vanilla ensemble.

2. Combination of TTA and weighted averaging

In evaluating the combination of TTA and weighted averaging, the analysis examines accuracy by varying the number of models, similar to the evaluation of TTA alone. Fig. 4.13 compares the accuracy achieved when combining weighted averaging and TTA with the accuracy obtained when using each method individually and when using none of them. Combining TTA, which is effective when the number of models is small, with weighted averaging, which is more effective when the number of models is large, consistently improved accuracy compared to the vanilla ensemble across all numbers of models.

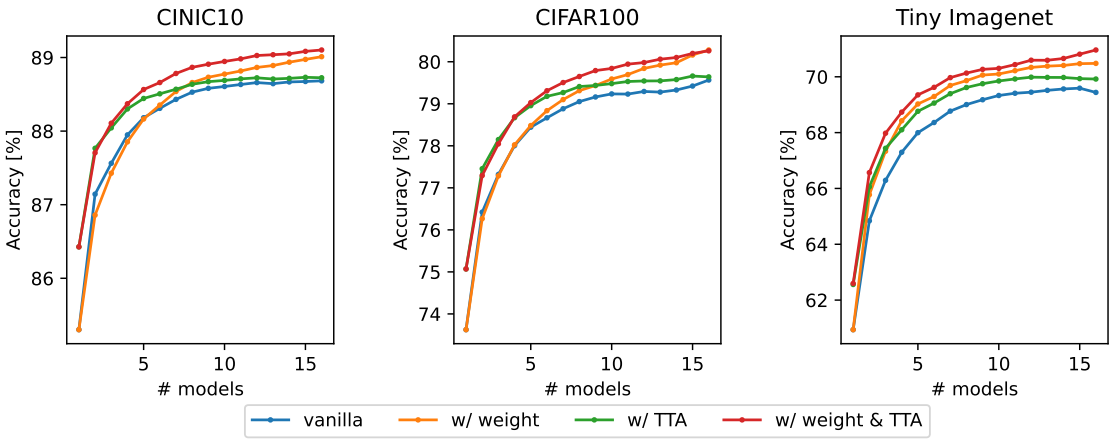

Fig. 4.13: Comparison of ensemble accuracies with TTA and weighted averaging and their combination in the HEE setting for varying numbers of base learners. The combination of TTA and weighted averaging resulted in the simultaneous effects of both techniques, resulting in consistently higher accuracy across all number of devices.

4.4.4 Discussion

Thus far, this work have assumed the context of Edge Ensembles and evaluated the performance by applying cascade, weighted averaging, and TTA to the model integration part through experiments.

For cascades, previous studies have reported that cascades reduce FLOPs and improve inference speed compared to simple ensembles [82]. However, our study assumes an Edge Ensemble where models on each device conduct inference in parallel. As a result, while the cascade reduced computational costs, it increased latency in our context. Furthermore, the m-parallel cascade mitigated the increased latency based on the parallelism factor m, allowing for the adjustment of the degree of computational costs reduction and the latency increase in the cascade.

Weighted averaging was the most effective when applied with the HEE configuration , similar to previous studies [77]. However, this study went one step further by evaluating performance, assuming variations in the availability of other devices in Edge Ensembles. The experimental results showed that weighted averaging is effective even when varying the number of base learners, demonstrating its effectiveness in the context of Edge Ensembles.

In the case of TTA, similar to findings from previous studies [61] that combined conventional DNN ensembles with TTA, the analysis shows that TTA on a single model

was the most effective, and its benefits diminished when used in ensembles. This study examined accuracy with TTA for different numbers of models to assess its utility in the context of Edge Ensembles. The results indicated that TTA serves as an option to improve the accuracy of Edge Ensembles when only a few nearby devices participate in collaborative inference.

In addition, the analysis validated the effectiveness of using ensemble prediction labels for learning TTA policies and weighting methods. The results show performance generally comparable to using ground truth labels. This outcome highlights the feasibility of using ensemble prediction labels alone to improve the accuracy of the ensemble itself without involving complex model training. However, it is important to note that this approach heavily relies on the accuracy of the ensemble prediction. For instance, if the Edge Ensembles contain numerous devices that produce random answers, the ensemble prediction result would be close to random, making effective learning impossible. Moreover, malicious models within the ensemble could lead to intentional misdirection of the predictions. Authors in [42] proposed removing devices with high error rates as a solution, but this approach also depends on the ensemble prediction results, remaining similar challenges. Addressing adversarial models in Edge Ensembles is a crucial future research direction.

4.5 Conclusion

This chapter leverages three approaches to enhance Edge Ensembles: cascade as an integration method with high efficiency, weighted averaging as an integration method to improve accuracy, and TTA as a method to integrate each base learner with improved accuracy. The experimental results support the effectiveness of these methods in improving Edge Ensembles' performance. The proposed m-parallel cascade addresses the latency issues present in conventional cascades for Edge Ensembles while reducing computational costs compared to simple ensembles. Weighted averaging improved accuracy, especially in the HEE setting, and enhanced accuracy in Edge Ensembles where the availability of neighboring devices varies. TTA significantly enhances the accuracy of single models compared to ensemble cases. This study also validate its effectiveness in scenarios with limited availability of neighboring devices for Edge Ensembles. Furthermore, our demonstration highlighted the effectiveness of ensemble label-based learning, enabling Edge Ensembles to adapt their model integration process on the edge side.

Additionally, this study evaluated the combination of these methods, specifically examining cascade with weighted averaging and TTA with weighted averaging. In both cases, the effects overlapped, confirming that employing multiple techniques together in edge

ensembles also yields effective results.

In conclusion, the collaborative inference system, which efficiently processes inference by leveraging the collective power of multiple devices, is expected to gain significant importance in the era of IoT. This study validated fundamental ensemble-based approaches in the context of collaborative inference while also introducing new ideas and assessing their effectiveness. However, critical security issues related to test data privacy and adversarial handling still remain for collaborative inference, necessitating further research in future work.

Chapter 5

Conclusion

(1) Summary

This dissertation proposes several novel approaches aimed at advancing Edge Ensembles, a collaborative inference framework particularly suited for ML applications in the rapidly expanding era of IoT, by focusing on ensemble methods designed for edge environments.

Chapter 3 introduced Extra Ferns, an edge-oriented fern-based ensemble learning method incorporating elements from Extra Trees and Random Ferns. It inherits the method of generating random nodes from Extra Trees, and it utilizes the fern structure from Random Ferns. To further enhance its performance, Extra Ferns optimizes node thresholds to improve accuracy and sparsifies leaf nodes to reduce memory usage. Experimental results demonstrate that Extra Ferns achieves higher accuracy than existing fern-based ensemble methods while significantly reducing memory accesses compared to traditional decision tree-based methods. Furthermore, Extra Ferns achieved additional accuracy improvements specifically for image data by incorporating random projection.

Chapter 4 examined the efficiency and accuracy improvements of model integration in Edge Ensembles utilizing DL. It adapted traditional model integration techniques to Edge Ensembles and validated their effectiveness. Additionally, it proposed two novel methods tailored for Edge Ensembles. The first method, the m-parallel cascade, extends the cascade approach by introducing the variable m, representing the number of models processed simultaneously. By adjusting m to meet the required latency, this approach minimized the number of models needed for the ensemble while maintaining latency constraints. The second proposal, a learning method based on ensemble prediction labels, utilized unlabeled data to learn the weights for weighted averaging and the transformation policies for TTA. This method improves the accuracy of model integration in Edge Ensembles without requiring labeled data.

In conclusion, this dissertation advances the field of Edge Ensembles by proposing an ensemble method based on lightweight Fern models and enhancing the accuracy and

efficiency of model integration in ensemble methods. The main contributions of this work are summarized as follows:

1. Extra Ferns achieved higher accuracy than existing fern-based bagging ensemble methods, contributing to the accuracy improvement of Edge Ensembles using ferns.

2. Extra Ferns significantly reduced memory access compared to existing lightweight decision tree ensemble learning methods, contributing to lower power consumption during training in edge environments.

3. This study applies various model integration methods to Edge Ensembles using deep learning and evaluates them under model settings tailored for Edge Ensembles, optimizing the integration process for both efficiency and accuracy.

4. This study proposes an optimization method for ensemble prediction integration using ensemble prediction labels, enabling Edge Ensembles to optimize predictions using only unlabeled inference data.

The first contribution made fern-based Edge Ensembles more practical for environments with extremely limited computational resources. The second contribution enabled these fern-based ensemble methods to perform training directly in edge environments, allowing for more adaptive and environment-specific inference. The third contribution enhanced the collaborative effects of DL-based Edge Ensembles, improving the overall performance of Edge Ensemble systems. Finally, the fourth contribution demonstrated the feasibility of optimizing model integration in edge environments by improving performance using only inference data. These contributions not only enhance the accuracy and efficiency of Edge Ensemble inference but also provide model training methods and analyses on collaborative frameworks, serving as valuable guidelines for designing practical cooperative inference systems. Through these contributions to Edge Ensembles, this study contributes to the advancement of edge AI in the emerging era of IoT.

(2) Future work

Finally, this dissertation concludes by discussing potential directions for future work. The collaborative inference system, which efficiently processes inference by leveraging the collective power of multiple devices, is expected to gain significant importance in the era of IoT. This study proposed lightweight ensemble learning methods tailored for collaborative inference in edge environments. It also validated fundamental ensemble-based approaches in the context of collaborative inference while introducing new ideas and assessing their

effectiveness. However, critical security issues related to test data privacy and adversarial handling remain for collaborative inference, necessitating further research in future work.

Recently, applications utilizing large language models and other generative models have emerged. While the execution of these models predominantly relies on large-scale servers with multiple GPUs operating in parallel, deploying such models on edge devices offers significant advantages in terms of real-time performance and privacy preservation. When these models are deployed on edge environments, distributed inference techniques in Edge Ensembles can potentially be leveraged to achieve load balancing, improved energy efficiency, and enhanced model redundancy in case of failures. Leveraging Edge Ensemble distributed inference techniques is expected to pave the way for novel applications and technological advancements in generative models.

Acknowledgements

I would like to express my heartfelt gratitude to the many individuals who provided immense support during the course of this research.

First and foremost, I am deeply grateful to my academic advisor, Prof. Masato Motomura, for granting me the invaluable opportunity to pursue PhD research in ArtIC. Over more than six years, since my time at Hokkaido University, he has continuously provided warm yet rigorous guidance. Whenever I faced obstacles in my research, his precise advice helped me overcome challenges, enabling me to complete this PhD dissertation. His dedication to research and thoughtful guidance have been influential for me. I will continue striving to follow his example.

I would also like to express my sincere gratitude to my former co-advisor, Prof. Jaehoon Yu, for his extensive support. He provided sharp insights and numerous helpful comments on the technical aspects of my research, which greatly contributed to my progress. In addition to research-related guidance, he taught me skills such as writing and preparing presentation materials. These skills are not only crucial for conducting research but also essential for succeeding as a professional in society. The knowledge and advice I gained from him have been invaluable and will remain a significant asset throughout my life. I am deeply thankful to him for his support and encouragement.

I am also indebted to Prof. Daichi Fujiki, Prof. Thiem Van Chu, Prof. Kazushi Kawamura and Prof. Kota Ando for their guidance. The valuable insights they shared during our research discussions were instrumental in advancing my work. I extend my heartfelt thanks to all of them.

Furthermore, I deeply appreciate the guidance from Prof. Tetsuya Asai and Prof. Shinya Takamaeda while I was part of LALSIE at Hokkaido University. Under their mentorship, I learned fundamental knowledge and methodologies for conducting research, which formed the foundation for my continued studies. Thank you very much for your invaluable support.

I am also grateful to my fellow students at LALSIE and ArtIC. The research discussions and daily interactions with them have been irreplaceable experiences for me. In particular, I would like to thank my peer, Mr. Junnosuke Suzuki, who has been a companion since our time at Hokkaido University and throughout our PhD studies. The encouragement and

camaraderie we shared were a great source of motivation, and I will always cherish those memories. I hope we can reunite for drinks after graduation! I would also like to thank my junior colleague, Mr. Yuta Nagahara, for bringing thoughtful gifts such as sake and "*Yadon*" udon at pivotal moments in my research. These gifts undoubtedly fueled me to persevere through the final PhD examinations. Thank you so much!

I am also deeply appreciative of Ms. Juri Hashimoto and Ms. Taeko Tsuchiya for their administrative and daily support. Their prompt and meticulous assistance with tasks such as managing research funds and preparing for trips allowed me to focus entirely on my research. Thank you sincerely for your support.

Finally, I would like to express my deepest gratitude to my parents, who have nurtured and supported me throughout my journey. Without their approval and encouragement to pursue a PhD program, I would not be where I am today. I am profoundly thankful for their unwavering support. Truly, thank you.

Bibliography

[1] L. D. Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[2] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[4] L. Stošić, M. Dimitrovska, and L. P. Stamenkova, "Exploring the expanding world of IoT: A comprehensive overview and considerations," *KNOWLEDGE – International Journal*, vol. 60, no. 3, pp. 359–363, 2023.

[5] S. Al-Sarawi, M. Anbar, R. Abdullah, and A. B. Al Hawari, "Internet of things market analysis forecasts, 2020-2030," in *Proceedings of World Conference on Smart Trends in Systems, Security and Sustainability*, pp. 449–453, 2020.

[6] N. Shlezinger, E. Farhan, H. Morgenstern, and Y. C. Eldar, "Collaborative inference via ensembles on the edge," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8478–8482, 2021.

[7] Y. Xu, Y. Wang, A. Zhou, W. Lin, and H. Xiong, "Deep neural network compression with single and multiple level quantization," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

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

[9] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proceedings of Advances in Neural Information Processing Systems*, pp. 1135–1143, 2015.

[10] V. Y. Kulkarni and P. K. Sinha, "Pruning of random forest classifiers: A survey and future directions," in *Proceedings of International Conference on Data Science & Engineering*, pp. 64–68, 2012.

[11] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015. 1503.02531.

[12] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, no. 6, pp. 1789–1819, 2021.

[13] H.-I. Liu, M. Galindo, H. Xie, L.-K. Wong, H.-H. Shuai, Y.-H. Li, and W.-H. Cheng, "Lightweight deep learning for resource-constrained environments: A survey," *ACM Computing Surveys*, vol. 56, no. 10, pp. 1–42, 2024.

[14] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017. arXiv:1704.04861.

[15] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

[16] B. Zoph and Q. Le, "Neural architecture search with reinforcement learning," in *Proceedings of International Conference on Learning Representations*, 2017.

[17] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *Journal of Machine Learning Research*, vol. 20, no. 55, pp. 1–21, 2019.

[18] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: Platform-aware neural architecture search for mobile," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 2815–2823, 2019.

[19] P. Dhilleswararao, S. Boppu, M. S. Manikandan, and L. R. Cenkeramaddi, "Efficient hardware architectures for accelerating deep neural networks: Survey," *IEEE Access*, vol. 10, pp. 131788–131828, 2022.

[20] R. Ormándi, I. Hegedűs, and M. Jelasity, "Gossip learning with linear models on fully distributed data," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 4, pp. 556–571, 2013.

[21] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of International Conference on Artificial Intelligence and Statistics*, vol. 54, pp. 1273–1282, 2017.

[22] J. Zhou, Y. Wang, K. Ota, and M. Dong, "AAIoT: Accelerating artificial intelligence in IoT systems," *IEEE Wireless Communications Letters*, vol. 8, no. 3, pp. 825–828, 2019.

[23] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "MoDNN: Local distributed mobile computing system for deep neural network," in *Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, pp. 1396–1401, 2017.

[24] N. Shlezinger and I. V. Bajić, "Collaborative inference for ai-empowered IoT devices," *IEEE Internet of Things Magazine*, vol. 5, no. 4, pp. 92–98, 2022.

[25] P. Joshi, M. Hasanuzzaman, C. Thapa, H. Afli, and T. Scully, "Enabling all in-edge deep learning: A literature review," *IEEE Access*, vol. 11, pp. 3431–3460, 2023.

[26] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge AI: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2020.

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

[28] W.-Q. Ren, Y.-B. Qu, C. Dong, Y.-Q. Jing, H. Sun, Q.-H. Wu, and S. Guo, "A survey on collaborative dnn inference for edge intelligence," *Machine Intelligence Research*, vol. 20, no. 3, pp. 370–395, 2023.

[29] M. Ganaie, M. Hu, A. Malik, M. Tanveer, and P. Suganthan, "Ensemble deep learning: A review," *Engineering Applications of Artificial Intelligence*, vol. 115, 2022.

[30] V. Pisetta, *New Insights into Decision Trees Ensembles*. PhD thesis, Université Lumière Lyon 2, 2012.

[31] T. G. Dietterich, "Ensemble methods in machine learning," in *Multiple Classifier Systems*, pp. 1–15, 2000.

[32] I. D. Mienye and Y. Sun, "A survey of ensemble learning: Concepts, algorithms, applications, and prospects," *IEEE Access*, vol. 10, pp. 99129–99149, 2022.

[33] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, 1996.

[34] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[35] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[36] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016.

[37] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "CatBoost: unbiased boosting with categorical features," in *Proceedings of Advances in Neural Information Processing Systems*, vol. 31, pp. 6639–6649, 2018.

[38] L. Breiman, "Stacked regressions," *Machine Learning*, vol. 24, no. 1, pp. 49–64, 1996.

[39] S. Hassan, A. Khosravi, and J. Jaafar, "Bayesian model averaging of load demand forecasts from neural network models," in *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, pp. 3192–3197, 2013.

[40] L. J. Gosink, C. C. Overall, S. M. Reehl, P. D. Whitney, D. L. Mobley, and N. A. Baker, "Bayesian model averaging for ensemble-based estimates of solvation-free energies," *The Journal of Physical Chemistry B*, vol. 121, no. 15, pp. 3458–3472, 2017.

[41] A. E. Raftery, T. Gneiting, F. Balabdaoui, and M. Polakowski, "Using bayesian model averaging to calibrate forecast ensembles," *Monthly Weather Review*, vol. 133, no. 5, pp. 1155–1174, 2005.

[42] X. Feng, C. Luo, B. Wei, J. Zhang, J. Li, H. Wang, W. Xu, M. C. Chan, and V. C. M. Leung, "Time-constrained ensemble sensing with heterogeneous IoT devices in intelligent transportation systems," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–12, 2022.

[43] M. Malka, E. Farhan, H. Morgenstern, and N. Shlezinger, "Decentralized low-latency collaborative inference via ensembles on the edge," 2022. arXiv:2206.03165.

[44] S. Rothe and D. Söffker, "Comparison of different information fusion methods using ensemble selection considering benchmark data," in *Proceedings of International Conference on Information Fusion*, pp. 73–78, 2016.

[45] T. Iqball and M. A. Wani, "Weighted ensemble model for image classification," *International Journal of Information Technology*, vol. 15, no. 2, pp. 557–564, 2023.

[46] M. J. van der Laan, E. C. Polley, and A. E. Hubbard, "Super learner," *Statistical Applications in Genetics and Molecular Biology*, vol. 6, no. 1, 2007.

[47] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. 511–518, 2001.

[48] L. Bourdev and J. Brandt, "Robust object detection via soft cascade," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 236–243, 2005.

[49] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes, "Ensemble selection from libraries of models," in *Proceedings of International Conference on Machine Learning*, 2004.

[50] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

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

[52] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger, "Snapshot ensembles: Train 1, get M for free," in *Proceedings of International Conference on Learning Representations*, 2017.

[53] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural Computation*, vol. 3, no. 1, pp. 79–87, 1991.

[54] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," in *Proceedings of International Conference on Learning Representations*, 2017.

[55] C. Riquelme, J. Puigcerver, B. Mustafa, M. Neumann, R. Jenatton, A. Susano Pinto, D. Keysers, and N. Houlsby, "Scaling vision with sparse mixture of experts," in *Proceedings of Advances in Neural Information Processing Systems*, pp. 8583–8595, 2021.

[56] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *Journal of Machine Learning Research*, vol. 23, no. 120, pp. 1–39, 2022.

[57] Y. Zhou, C. Guo, X. Wang, Y. Chang, and Y. Wu, "A survey on data augmentation in large model era," 2024. arXiv:2401.15422.

[58] Y. Lou and M. Obukhov, "BDT: Gradient boosted decision tables for high accuracy and scoring efficiency," in *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1893–1901, 2017.

[59] A. Gurianov, "Oblivious piecewise-linear decision trees," in *Proceedings of International Conference on Information Technology and Nanotechnology*, pp. 1–5, 2022.

[60] M. Ozuysal, P. Fua, and V. Lepetit, "Fast keypoint recognition in ten lines of code," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1–8, 2007.

[61] A. Ashukha, A. Lyzhov, D. Molchanov, and D. Vetrov, "Pitfalls of in-domain uncertainty estimation and ensembling in deep learning," in *Proceedings of International Conference on Learning Representations*, 2020.

[62] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, Oct. 2001.

[63] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, pp. 3–42, Mar. 2006.

[64] M. B. Kursa, "rFerns: An implementation of the random ferns method for general-purpose machine learning," *Journal of Statistical Software*, vol. 61, pp. 1–13, Nov. 2014.

[65] E. Bingham and H. Mannila, "Random projection in dimensionality reduction: Applications to image and text data," in *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 245–250, 2001.

[66] P. Li, T. J. Hastie, and K. W. Church, "Very sparse random projections," in *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 287–296, 2006.

[67] G. Louppe, *Understanding random forests: From theory to practice*. PhD thesis, University of Liège, 2014.

[68] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *Proceedings of IEEE International Solid-State Circuits Conference*, pp. 10–14, 2014.

[69] D. Dua and C. Graff, "UCI machine learning repository." `http://archive.ics.uci.edu/ml`, 2017.

[70] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms." arXiv:1708.07747, 2017.

[71] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, "Deep learning for classical japanese literature." arXiv:1812.01718, 2018.

[72] S. Acharya, A. K. Pant, and P. K. Gyawali, "Deep learning based large scale handwritten Devanagari character recognition," in *Proceedings of the International Conference on Software, Knowledge Information, Industrial Management and Applications*, pp. 1–6, 2015.

[73] P. Baldi, P. Sadowski, and D. Whiteson, "Searching for exotic particles in high-energy physics with deep learning," *Nature Communications*, vol. 5, p. 4308, July 2014.

[74] S. F. Yilmaz, B. Hasırcıoğlu, and D. Gündüz, "Over-the-air ensemble inference with model privacy," in *Proceedings of IEEE International Symposium on Information Theory*, pp. 1265–1270, 2022.

[75] Y. Yang, H. Lv, and N. Chen, "A survey on ensemble learning under the era of deep learning," *Artificial Intelligence Review*, vol. 56, pp. 5545–5589, 2023.

[76] O. Sagi and L. Rokach, "Ensemble learning: A survey," *WIREs Data Mining and Knowledge Discovery*, vol. 8, no. 4, 2018.

[77] C. Ju, A. Bibaut, and M. J. van der Laan, "The relative performance of ensemble methods with deep convolutional neural networks for image classification," *Journal of Applied Statistics*, vol. 45, no. 15, pp. 2800–2818, 2018.

[78] N. Du, Y. Huang, A. M. Dai, S. Tong, D. Lepikhin, Y. Xu, M. Krikun, Y. Zhou, A. W. Yu, O. Firat, B. Zoph, L. Fedus, M. P. Bosma, Z. Zhou, T. Wang, E. Wang, K. Webster, M. Pellat, K. Robinson, K. Meier-Hellstern, T. Duke, L. Dixon, K. Zhang, Q. Le, Y. Wu, Z. Chen, and C. Cui, "GLaM: Efficient scaling of language models with mixture-of-experts," in *Proceedings of International Conference on Machine Learning*, vol. 162, pp. 5547–5569, 2022.

[79] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," in *Proceedings of International Conference on Learning Representations*, 2017.

[80] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *Proceedings of European Conference on Computer Vision*, pp. 646–661, 2016.

[81] Z.-H. Zhou, J. Wu, and W. Tang, "Ensembling neural networks: Many could be better than all," *Artificial Intelligence*, vol. 137, no. 1, pp. 239–263, 2002.

[82] X. Wang, D. Kondratyuk, E. Christiansen, K. M. Kitani, Y. Alon, and E. Eban, "Wisdom of committees: An overlooked approach to faster and more accurate models," in *Proceedings of International Conference on Learning Representations*, 2022.

[83] C. Ju, M. Combs, S. D. Lendle, J. M. Franklin, R. Wyss, S. Schneeweiss, and M. J. van der Laan, "Propensity score prediction for electronic healthcare databases using super learner and high-dimensional propensity score methods," *Journal of Applied Statistics*, vol. 46, no. 12, pp. 2216–2236, 2019.

[84] A. Lyzhov, Y. Molchanova, A. Ashukha, D. Molchanov, and D. Vetrov, "Greedy policy search: A simple baseline for learnable test-time augmentation," in *Proceedings of Conference on Uncertainty in Artificial Intelligence*, vol. 124, pp. 1308–1317, 2020.

[85] D. Shanmugam, D. Blalock, G. Balakrishnan, and J. Guttag, "Better aggregation in test-time augmentation," in *Proceedings of IEEE International Conference on Computer Vision*, pp. 1194–1203, 2021.

[86] I. Kim, Y. Kim, and S. Kim, "Learning loss for test-time augmentation," in *Proceedings of Advances in Neural Information Processing Systems*, pp. 4163–4174, 2020.

[87] E. D. Cubuk, B. Zoph, J. Shlens, and Q. Le, "Randaugment: Practical automated data augmentation with a reduced search space," in *Proceedings of Advances in Neural Information Processing Systems*, pp. 18613–18624, 2020.

[88] C. Luo, X. He, J. Zhan, L. Wang, W. Gao, and J. Dai, "Comparison and benchmarking of ai models and frameworks on mobile devices," 2020. arXiv:2005.05085.

[89] M. Ghamari, H. Arora, R. S. Sherratt, and W. Harwin, "Comparison of low-power wireless communication technologies for wearable health-monitoring applications," in *Proceedings of International Conference on Computer, Communications, and Control Technology*, pp. 1–6, 2015.

List of Publications

Journal Papers

1. **S. Kumazawa,** K. Kawamura, T. Van Chu, M. Motomura and J. Yu, "ExtraFerns: Fully Parallel Ensemble Learning Technique with Random Projection and Non-Greedy yet Minimal Memory Access Training," *International Journal of Networking and Computing*, Vol. 11, No. 2, pp. 215–230, Jul. 2021.

2. **S. Kumazawa,** J. Yu, K. Kawamura, T. V. Chu and M. Motomura, "Toward Improving Ensemble-Based Collaborative Inference at the Edge," *IEEE Access*, Vol. 12, pp. 6926–6940, Jan. 2024

International Conference Proceedings

1. **S. Kumazawa,** K. Kawamura, T. Van Chu, M. Motomura and J. Yu, "ExtraFerns: Fully Parallel Ensemble Learning Technique with Non-Greedy yet Minimal Memory Access Training," *2020 Eighth International Symposium on Computing and Networking (CANDAR)*, Naha, Japan, Nov. 2020, pp. 146–152.

Journal Papers (Coauthored)

1. J. Suzuki, J. Yu, M. Yasunaga, Á. López García-Arias, Y. Okoshi, **S. Kumazawa**, K. Ando, K. Kawamura, T. Van Chu, and M. Motomura, "Pianissimo: A Sub-mW Class DNN Accelerator With Progressively Adjustable Bit-Precision," *IEEE Access*, vol. 12, pp. 2057–2073, Jan. 2024,

International Conference Proceedings (Coauthored)

1. T. Van Chu, Y. Mizutani, Y. Nagahara, **S. Kumazawa,** K. Kawamura, J. Yu, and M. Motomura, "Decision Forest Training Accelerator Based on Binary Feature Decomposition," *2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Marina Del Rey, CA, USA, May. 2023. p. 215

2. J. Suzuki, J. Yu, M. Yasunaga, Á. López García-Arias, Y. Okoshi, **S. Kumazawa**, K. Ando, K. Kawamura, T. Van Chu, and M. Motomura, "Pianissimo: A Sub-mW Class DNN Accelerator with Progressive Bit-by-Bit Datapath Architecture for Adaptive Inference at Edge," *2023 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, Kyoto, Japan, Jun. 2023. pp. 1–2,

3. M. Watanabe, **S. Kumazawa**, T. Van Chu, K. Kawamura, J. Yu and M. Motomura, "Exploration of Hyperdimensional Computing Using Locality-Sensitive Hashing Mechanism on FPGA," *2024 IEEE International Conference on Consumer Electronics (ICCE)*, Las Vegas, NV, USA, Jan. 2024. pp. 1–2,