



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DIPARTIMENTO DI INFORMATICA - SCIENZA E INGEGNERIA
Corso di Laurea Magistrale in Ingegneria Informatica

Federated Fine-Tuning of Mixture of Experts Vision Transformers

Tesi di Laurea Magistrale
in
Mobile Systems

Relatore:
Chiar.mo Prof. Ing. Paolo Bellavista

Candidato:
Filippo Serafini

Correlatori:
Alessio Mora
Lorenzo Sani

Sessione I
Anno Accademico 2025/2026

Abstract

Mixture-of-Experts (MoE) architectures decouple a model’s representational capacity from its per-token computational cost, activating only a small subset of specialised experts for each input. Their primary advantage lies in the ability to achieve significantly greater representational capacity than their dense counterparts, while maintaining the same activation costs. In parallel, Federated Learning (FL) enables distributed training directly on edge devices—periodically aggregating their local updates—without centralising raw data, thereby ensuring compliance with strict privacy regulations and unlocking a vast wealth of information distributed across the edge. However, integrating the two poses a systemic challenge: cross-device FL is bound by strict bandwidth and memory constraints, further complicated by the statistical heterogeneity (non-IID) of local datasets. Yet the inherent sparsity of MoEs can be exploited to overcome these barriers. Adapting the *partial-replica* paradigm of the FoMoE framework—each client hosting only a subset of the experts—this thesis investigates the federated fine-tuning of a Vision Mixture-of-Experts (V-MoE-S/32) on edge devices holding severely non-IID data, combined with Low-Rank Adaptation (LoRA) and a Sparsity-aware Model Aggregation (SaMA) scheme. Under a *static* expert assignment, restricting each client to half—or fewer—of the expert pool costs only one to two percentage points of Top-1 accuracy with respect to the full-replica baseline, while reducing the per-round communication payload by up to 28% and the static on-device memory by up to 36%. This static assignment, however, creates a mismatch between the experts a client needs most and those it was initially allocated. To close this gap, this work finally designs and evaluates a novel server-orchestrated *dynamic expert reassignment* protocol, which periodically migrates experts towards the clients that demand them most. We find that, rather than raising the accuracy ceiling, it mainly *stabilises* the federation against sub-optimal, static initial placements, at only a marginal communication overhead of one to three percent of the total traffic. Taken together, the results indicate that partial expert replication, parameter-efficient adaptation, and sparsity-aware aggregation compose into a practical recipe for bringing sparse vision models within the memory, compute, and bandwidth constraints of real-world federated edge networks. More broadly, by decoupling local hardware limits from the overall model size, this design enables edge networks to train significantly larger and more capable architectures.

Contents

1	Introduction	1
1.1	Context and Motivation	2
1.2	Problem Statement	3
1.3	Objectives and Research Questions	3
1.4	Methodology	5
1.5	Thesis Structure	5
2	Background	7
2.1	Machine Learning	9
2.1.1	ML Training	9
2.1.2	Data Preparation	9
2.1.3	Data Partitioning	10
2.2	Deep Learning	10
2.2.1	Model Initialisation	10
2.2.2	Forward Pass	11
2.2.3	Loss Function	11
2.2.4	Backward Pass	12
2.2.5	Optimisation Algorithm	12
2.2.6	Learning Rate	13
2.2.7	Epochs and Batches	13
2.2.8	Validation and Evaluation	13
2.2.9	Hyperparameter Tuning	14
2.2.10	Training, Fine-Tuning and Inference	14
2.3	Federated Learning	14
2.3.1	FL Setting	14
2.3.2	Advantages and Challenges	15
2.3.3	Cross-Silo and Cross-Device	16
2.3.4	Heterogeneity in Federated Learning	16
2.4	Mixture of Experts Architecture	17
2.4.1	Scaling Laws and Model Capacity	17
2.4.2	Sparsity and Conditional Computation	17
2.4.3	MoE Formulation	18
2.4.4	Routing Challenges and Load Balancing	19
2.4.5	Architectural Trade-offs	20
2.5	Vision Transformers	20
2.5.1	From Convolutions to Patch Sequences	20

2.5.2	Architecture Overview	20
2.5.3	Scaling and Performance	22
2.6	Vision Mixture-of-Experts	22
2.6.1	The MoE Block and Expert Architecture	22
2.6.2	Expert Capacity and Token Dropping	24
2.6.3	Scaling Efficiency and Performance	24
3	Related Work	26
3.1	Partial Expert Replication in Distributed MoE Training	27
3.1.1	Problem Statement	27
3.1.2	The Partial-Replica Paradigm	28
3.1.3	Expert Assignment Strategies	30
3.1.4	Evaluation and Results	30
3.1.5	Relevance and Implications for the Thesis	31
3.2	Parameter-Efficient Fine-Tuning	32
3.2.1	Low-Rank Decomposition	32
3.2.2	Architectural Advantages	32
3.2.3	Relevance and Implications for the Thesis	33
3.3	Sparsity-aware Aggregation in FL	33
3.3.1	FedAvg and its Limitations	34
3.3.2	Sparsity-aware Model Aggregation (SaMA)	34
3.3.3	Relevance and Implications for the Thesis	35
4	Methodology: Partial-Replica V-MoE in Federated Learning	36
4.1	Objectives and Contributions	37
4.1.1	Partial-Replica Paradigm in FL	37
4.1.2	Static Expert Assignment	38
4.1.3	Dynamic Expert Reassignment and Trade-off Exploration	38
4.2	Architectural Analysis for Edge Deployment	39
4.3	Fine-Tuning in Edge Federated Learning	41
4.3.1	The PEFT Paradigm	41
4.4	Model Selection: V-MoE Small/32	41
4.4.1	Choice Rationale	42
4.4.2	Architectural Overview	44
4.5	Fine-Tuning Setting	45
4.6	Sparsity-aware Model Aggregation	46
4.6.1	Client-Side: Token Tracking Mechanism	46
4.6.2	Server-Side: Dual-Track Aggregation	47
4.7	Partial-Replica Setup	48
4.7.1	Static Assignment	48
4.7.2	Practical Deployment	49
4.8	Dynamic Expert Assignment	51
4.8.1	Methodology	51
4.8.2	Swap Rules	52
4.8.3	Swap Evaluation Metric	53
4.8.4	Reassignment Algorithm	55

5	Experimental Results	58
5.1	Experimental Setup	60
5.1.1	Flower Framework	60
5.1.2	Federated Configuration	61
5.1.3	Datasets and Non-IID Partitioning	62
5.2	Evaluation Baselines	65
5.2.1	Centralised V-MoE Upper Bound	65
5.2.2	Full-Replica Federated V-MoE	67
5.2.3	Dense Federated ViT-S/32	67
5.2.4	Results	67
5.3	Partial-Replica: Static and Dynamic	69
5.3.1	Premises and Expectations	69
5.3.2	Replication Strategies at Initialisation	70
5.3.3	Expected Outcomes	71
5.4	Static Assignment	71
5.4.1	Model Performance	71
5.4.2	Expert Specialisation	72
5.4.3	Ghost Activations	76
5.4.4	System Metrics	78
5.4.5	The accuracy–system trade-off	80
5.5	Dynamic Reassignment	81
5.5.1	Model Performance	81
5.5.2	Expert Specialisation	82
5.5.3	Ghost Activations	84
5.5.4	Swaps Analysis	85
5.5.5	The Cost of Dynamism	86
5.6	Summary of the Findings	90
5.7	Conclusions and Future Work	93
	Bibliography	96

List of Figures

2.1	Round in a Federated Learning setting.	15
2.2	Routing mechanism in MoE. Top-2 selection.	19
2.3	Vision Transformer architecture.	21
2.4	V-MoE structure.	23
2.5	MLP structure	23
3.1	Comparison between a transformer MoE block as a full versus partial replica	29
3.2	LoRA reparametrisation	33
4.1	Dense transformer block	44
4.2	Sparse transformer block	45
5.1	Flower logo	60
5.2	CIFAR-100 image examples	63
5.3	Office-Home domains example	63
5.4	Domain shift partitioning on Office-Home	65
5.5	Pachinko-thematic data partitioning	66
5.6	Server-side top-1 expert token routing on CIFAR-100.	73
5.7	Server-side top-1 expert token routing on Office-Home.	75
5.8	Top-1 ghost activation rate for the $E_{\text{loc}} = 3$ static partial-replica.	76
5.9	Top-1 ghost activation rate for the $E_{\text{loc}} = 4$ static partial-replica.	77
5.10	Server-side top-1 expert token routing on CIFAR-100 under dynamic reassignment.	83
5.11	Server-side top-1 expert token routing on Office-Home under dynamic reassignment.	83
5.12	Server-side top-1 ghost rate under dynamic reassignment.	84
5.13	Per-round communication payload under dynamic reassignment.	89
5.14	Server-side Acc@1 convergence: full-replica versus static and dynamic partial replicas.	91
5.15	Relative system savings versus accuracy cost of partial replication.	92
5.16	Per-seed final accuracy: static versus dynamic reassignment.	93

List of Tables

4.1	Parameter budget comparison: V-MoE-S/32 versus V-MoE-B/16. . .	42
4.2	Empirical hardware profiling under full fine-tuning.	43
4.3	Parameter breakdown and communication payload for the V-MoE-S/32.	46
5.1	Baseline results on the two benchmarks	68
5.2	Capacity and active-compute comparison: dense ViT-S/32 versus V-MoE-S/32.	69
5.3	Static partial-replica accuracy on CIFAR-100 and Office-Home. . . .	72
5.4	Communication payload reduction under static partial-replica allocation.	79
5.5	System profiling of the static partial-replica V-MoE.	79
5.6	Dynamic and static partial-replica accuracy on CIFAR-100 and Office-Home.	82
5.7	Swap dynamics of the partial-dynamic reassignment on both benchmarks.	85
5.8	Server-side demand versus client-side replication on Office-Home. . . .	86
5.9	Measured payload units of the ordinary round and of one expert migration.	88
5.10	Communication overhead of dynamic reassignment over the static baseline.	90

Chapter 1

Introduction

Contents

1.1	Context and Motivation	2
1.2	Problem Statement	3
1.3	Objectives and Research Questions	3
1.4	Methodology	5
1.5	Thesis Structure	5

1.1 Context and Motivation

Over the last decade, the progress of deep learning has been driven largely by scale: empirical scaling laws [1] consistently reward larger models with better representational capacity and generalisation. Yet this trajectory collides with two independent constraints that define an increasing share of real-world deployments.

The first constraint concerns *where the data lives*. Modern data is generated at the edge—on smartphones, cameras, vehicles, and IoT sensors—and is often personal, proprietary, or regulated. Centralising it for training is frequently undesirable or forbidden. Federated Learning (FL) [2] addresses this problem by inverting the paradigm: instead of moving the data to the model, the model is trained collaboratively across the data holders, and only parameter updates are exchanged with a central aggregator. In its most challenging scenario—*cross-device* FL—the participants are numerous, unreliable edge devices with strict limits on memory, compute, energy, and network bandwidth.

The second constraint concerns *how models scale*. Dense architectures activate every parameter for every input, so their computational cost grows proportionally with their capacity. Sparse *Mixture-of-Experts* (MoE) architectures [3, 4] break this proportionality: they replace monolithic feed-forward layers with pools of specialised *experts*, of which a learned router activates only a small subset (typically two) per token. The result is a model whose knowledge capacity can grow substantially while its per-token execution cost remains that of a much smaller network. This principle of conditional computation has been successfully carried into computer vision by the Vision Mixture-of-Experts (V-MoE) [5], a sparse variant of the Vision Transformer [6].

Taken together, these two paradigms appear naturally complementary: sparse models promise datacentre-level capacity at edge-level compute, exactly what a federation of constrained devices needs. In practice, however, their combination exposes a structural incompatibility. The computational sparsity of an MoE does not translate into *memory* or *communication* sparsity: since the router may dispatch any token to any expert, the **entire** expert pool must reside in each device’s memory, and in a federated setting the complete set of expert updates must be exchanged at every communication round. Every client is thus forced to behave as a full replica of the global model. This *full-replica barrier* ties the per-client memory footprint and communication payload to the total model size—precisely the two resources that cross-device FL cannot afford. On edge hardware, the consequence is not merely inefficiency: a device whose VRAM cannot accommodate the full expert pool is structurally excluded from the federation, biasing the global model towards the data of high-tier hardware.

Recent work has begun to dismantle this barrier in a different context. The Fo-MoE framework [7] demonstrated that, when pre-training Large Language Models across weakly connected datacentres, each worker can host only a *partial replica* of the expert pool: tokens routed to locally absent experts (*ghost experts*) are simply *skipped*, bypassing the missing computation instead of transferring activations across the network. This co-design of training strategy and architectural sparsity substantially reduces both memory and communication costs with a negligible im-

pact on model quality. Whether these benefits survive the transition to a radically different regime—the *fine-tuning* of a pre-trained vision model, on a federation of heterogeneous edge devices, over severely non-IID data—is an open question, and the central subject of this thesis.

1.2 Problem Statement

This thesis targets the federated fine-tuning of a pre-trained Vision Mixture-of-Experts across resource-constrained edge devices. Concretely, the problem decomposes into three coupled costs, each of which is prohibitive if addressed naively:

The cost of distributing the model. Replicating the full expert pool on every client saturates the memory of constrained devices and inflates the per-round communication payload. The partial-replica paradigm removes this cost, but introduces its own tension: whenever a client’s router requests an expert the client does not host, the skip-token mechanism silently discards that computation. Under non-IID data—where local routing demand is highly skewed—this can concentrate the damage on precisely the experts a client needs most.

The cost of adapting the model. Full fine-tuning of a multi-million-parameter backbone requires storing gradients and optimiser states that triple the memory footprint of the trainable weights, and uploading full-model updates at every round. Parameter-Efficient Fine-Tuning, in the form of Low-Rank Adaptation (LoRA) [8], freezes the pre-trained backbone and learns only small low-rank residual matrices, collapsing both costs by orders of magnitude.

The cost of aggregating sparse updates. Standard FedAvg [2] weights every parameter of a client’s update by its dataset size, ignoring the fact that sparse routing exposes different experts to vastly different amounts of data—including none at all. Blindly averaging an expert across clients that never activated it dilutes the specialised knowledge of those that did. A sparsity-aware aggregation scheme [9] that weights each expert by its actual token utilisation is required to preserve update integrity.

A further dimension cuts across all three: *statistical heterogeneity*. Fine-tuning implies that the network’s routers start with strong, pre-trained token-to-expert affinities. When the local data is non-IID, these frozen preferences clash with the local distribution, producing severely imbalanced expert demand that a static expert placement cannot anticipate. This raises the final question addressed by this work: whether the expert-to-client topology should remain fixed, or instead *adapted dynamically* to the demand that clients express during training.

1.3 Objectives and Research Questions

The primary objective of this thesis is to bridge the gap between the representational power of V-MoE architectures and the hardware and bandwidth constraints

of real-world, cross-device Federated Learning. To this end, the work extracts the partial-replica paradigm from its original datacentre pre-training context and adapts it for federated fine-tuning on the edge. This broad objective breaks down into three research questions, which structure the entire experimental investigation:

- **RQ1 — At what cost does partial replication come?** When each client hosts only a subset of the expert pool and skips the computations it cannot perform, how much accuracy is lost with respect to a full replica trained on the same non-IID partitions—and how much memory, bandwidth, and compute is saved in exchange?
- **RQ2 — How does a pre-trained router react to missing experts?** Fine-tuning inherits a gating network with strong, pre-trained token-to-expert affinities. Does the router learn to redirect its requests towards the experts a client actually hosts, or does it keep demanding its preferred experts regardless of their local availability?
- **RQ3 — Should the expert-to-client assignment adapt during training?** If the router will not adapt to the local infrastructure, can the infrastructure adapt to the router? Specifically, can a server-orchestrated reassignment protocol—migrating experts towards the clients that demand them most—improve accuracy or stability enough to justify its additional communication cost?

In answering these questions, the thesis makes the following contributions:

- **An end-to-end partial-replica federated fine-tuning system.** A complete FL pipeline—built on the Flower framework [10]—in which each client hosts only $E_{\text{loc}} < E_{\text{tot}}$ experts per MoE layer of a pre-trained V-MoE-S/32, adapts them through LoRA modules, and relies on the ghost-expert/skip-token mechanism to bypass locally missing computations. Expert updates are aggregated with a Sparsity-aware Model Aggregation (SaMA) scheme.
- **A systematic study of static expert assignment under non-IID data.** We test two ways (balanced and random) to assign experts to clients at the beginning. We give each client a local budget of either 3 or 4 experts out of 8 total ($E_{\text{loc}} \in \{3, 4\}$ out of $E_{\text{tot}} = 8$). We evaluate this on two datasets with different types of unbalanced data: label skew on CIFAR-100 and domain skew on Office-Home. Finally, we explain why using only a partial set of experts lowers accuracy. We do this by looking at the router’s internal behavior: how experts specialize, how often *ghost activations* occur, and how inflexible the initial routing rules are.
- **A dynamic expert reassignment protocol.** A novel server-orchestrated mechanism that periodically migrates experts towards the clients demanding them most, combining temporally smoothed demand scores with capacity-preserving swap rules.. The protocol is evaluated in a strictly paired comparison against its static counterpart, asking whether its benefits justify its costs.

- **A full system-level characterisation.** Beyond accuracy, every configuration is profiled along the axes that gate real edge deployments—communication payload, static and peak memory, and per-image compute—quantifying the trade-off that partial replication offers between a marginal accuracy sacrifice and substantial resource savings.

1.4 Methodology

The experimental platform is a simulated cross-device federation of eight clients, built on the Flower framework [10]. The global model is a V-MoE-S/32 pre-trained on ImageNet-1k: a Vision Transformer of the *Small* family whose last two odd blocks replace the feed-forward layer with a pool of eight experts, of which a learned router activates the top-two per token. The choice of this backbone is itself hardware-driven: among the publicly released V-MoE checkpoints, it is the one whose footprint is compatible with the memory budget of edge-class devices.

Three building blocks, each borrowed from a distinct line of research and adapted to our setting, compose the system. First, the *partial replica*: each client permanently hosts only three or four of the eight experts per MoE layer, and tokens routed to a locally absent expert are skipped rather than transferred. Second, *Low-Rank Adaptation*: the pre-trained backbone stays frozen, and only small low-rank residual matrices—together with a handful of lightweight modules—are trained and exchanged, confining the communication payload to a few percent of the network. Third, *sparsity-aware aggregation*: the server weights each expert’s update by the number of tokens each client actually routed to it, protecting rarely-hosted experts from being averaged into irrelevance.

Statistical heterogeneity is induced through two complementary non-IID regimes. On CIFAR-100, a hierarchical partitioning concentrates semantically related super-classes on each client, producing a severe *label skew*; on Office-Home, the four visual domains of the dataset are distributed across the federation, producing a *domain skew*.

The evaluation proceeds along three axes: task accuracy, the router’s internal behaviour, observed through expert specialisation and ghost activation rates, and system metrics. Each configuration is repeated over three independent expert-placement seeds, and the dynamic reassignment protocol is compared against the static assignment on the *same* seeds, so that any difference reflects the mechanism alone.

1.5 Thesis Structure

The remainder of this thesis is organised as follows:

Chapter 2 introduces the theoretical foundations of the work: the machine learning and deep learning fundamentals, the Federated Learning paradigm and its heterogeneity challenges, the Mixture-of-Experts architecture with its routing and load-balancing mechanisms, and the Vision Transformer and V-MoE architectures on which the system is built.

Chapter 3 reviews the three foundational lines of research this thesis combines: the FoMoE framework for partial expert replication in distributed MoE training, Low-Rank Adaptation for parameter-efficient fine-tuning, and the sparsity-aware aggregation scheme of HFedMoE, discussing for each its relevance and the aspects adapted to our setting.

Chapter 4 presents the methodology of the proposed system: the research objectives, the hardware-driven selection of the V-MoE-S/32 backbone, the LoRA-based fine-tuning configuration, the implementation of SaMA, the static partial-replica setup with its ghost-expert routing, and the design of the dynamic expert reassignment protocol.

Chapter 5 reports the experimental evaluation: the federated configuration and the non-IID partitioning strategies, the baseline results bracketing the study, and the in-depth analysis of the static and dynamic partial-replica regimes across accuracy, routing behaviour, and system metrics. The chapter closes with the conclusions of the study and the directions it opens for future work.

Chapter 2

Background

Contents

2.1	Machine Learning	9
2.1.1	ML Training	9
2.1.2	Data Preparation	9
2.1.3	Data Partitioning	10
2.2	Deep Learning	10
2.2.1	Model Initialisation	10
2.2.2	Forward Pass	11
2.2.3	Loss Function	11
2.2.4	Backward Pass	12
2.2.5	Optimisation Algorithm	12
2.2.6	Learning Rate	13
2.2.7	Epochs and Batches	13
2.2.8	Validation and Evaluation	13
2.2.9	Hyperparameter Tuning	14
2.2.10	Training, Fine-Tuning and Inference	14
2.3	Federated Learning	14
2.3.1	FL Setting	14
2.3.2	Advantages and Challenges	15
2.3.3	Cross-Silo and Cross-Device	16
2.3.4	Heterogeneity in Federated Learning	16
2.4	Mixture of Experts Architecture	17
2.4.1	Scaling Laws and Model Capacity	17
2.4.2	Sparsity and Conditional Computation	17
2.4.3	MoE Formulation	18
2.4.4	Routing Challenges and Load Balancing	19
2.4.5	Architectural Trade-offs	20

2. Background

2.5	Vision Transformers	20
2.5.1	From Convolutions to Patch Sequences	20
2.5.2	Architecture Overview	20
2.5.3	Scaling and Performance	22
2.6	Vision Mixture-of-Experts	22
2.6.1	The MoE Block and Expert Architecture	22
2.6.2	Expert Capacity and Token Dropping	24
2.6.3	Scaling Efficiency and Performance	24

This chapter introduces the theoretical background and core concepts needed to understand the rest of the thesis. It defines key terminology and provides the foundation necessary to comprehend the methodologies, design choices, and contributions presented in this work.

2.1 Machine Learning

Machine Learning (ML) constitutes a foundational paradigm within the broader field of Artificial Intelligence (AI). It focuses on statistical algorithms that can learn from data and, subsequently, make predictions. By shifting the focus from rule-based programming to data-driven pattern recognition, ML has become instrumental in solving complex, non-linear problems across various domains.

The core mechanism through which a machine learning algorithm acquires knowledge is referred to as the training phase.

2.1.1 ML Training

Training is an iterative optimisation process where the model is exposed to a subset of data, termed the training set, allowing it to adjust its internal parameters to accurately capture the underlying data distribution and latent patterns. However, since raw data is rarely acquired in a compute-ready format, a rigorous preprocessing phase is an essential prerequisite.

2.1.2 Data Preparation

In ML training, data preparation involves several subtasks designed to transform raw information into a structured, high-quality format suitable for processing.

Data collection It is the initial phase of gathering raw observations from diverse and potentially disparate sources. The volume, quality, and variety of the resulting dataset is fundamental for setting the model’s upper-bound performance. Data sources vary according to the type of task at hand: they can involve scraping text documents from the web or aggregating X-ray scans from hospital archives.

Data preprocessing This step focuses on cleaning and transforming the raw data into a numerically stable input for the training process. This includes removing any kind of noise, resolving inconsistencies, and handling missing values. Furthermore, it involves structural mathematical transformations (e.g., tokenising text into integer IDs or normalising image pixel values from $[0, 255]$ to a $[0, 1]$ range).

Data splitting To properly evaluate a model’s capability to generalise to unseen data, the preprocessed dataset is divided into three distinct, non-overlapping subsets, usually following a standard 80% – 10% – 10% strategy:

- *Training set*, used to update the model weights.

2. Background

- *Validation set*, used for evaluating the model’s performance throughout training and for the hyperparameter tuning.
- *Test set*, used to assess the final performance and generalisation capability of the trained model.

2.1.3 Data Partitioning

When accounting for Distributed Machine Learning and Federated Learning scenarios, the global dataset needs to be divided among multiple physical nodes (e.g., edge devices, servers, etc.). Data partitioning enables collaborative and parallel model training across distributed environments while minimising the need for centralised data aggregation. It can occur along different axes.

Horizontal partitioning Also known as sample-based partitioning, it consists of dividing the dataset by rows. Each node holds a different subset of data samples, but the feature space (the type of data collected) is identical across all nodes.

Vertical partitioning Also known as feature-based partitioning, this involves dividing the dataset by columns. In this scenario, multiple nodes possess data belonging to the exact same entities (e.g., the same users), but each node holds a different subset of features.

Hybrid partitioning This is a real-world scenario that combines horizontal and vertical partitioning methods. Nodes only share a partial overlap in both the sample they observe and the features they collect.

2.2 Deep Learning

Deep Learning (DL) is a formalised subfield of machine learning that utilises multi-layered Artificial Neural Networks (ANNs) to solve complex problems. The term “deep” refers to the depth of these networks: the many successive layers of interconnected neurons that allow the model to progressively extract increasingly abstract features from data.

2.2.1 Model Initialisation

The first step consists of defining the model design and architecture, followed by the initialisation of the parameters to suitable values.

Architecture design The model’s architecture typically specifies the number and type of layers (e.g., fully-connected, convolutional, recurrent), the number of neurons per layer (known as the layer’s width), how the layers are connected, and the activation functions used at each stage.

Parameter initialisation Once the architecture is set, the model parameters (weights and biases) are initialised before training begins. This step is crucial, since proper initial values help prevent issues such as vanishing or exploding gradients and promote stable convergence.

Weights are typically initialised with small random values drawn from a distribution (e.g., uniform or normal distribution), while biases are set to zero or small constants. Common initialisation techniques involve Xavier/Glorot initialisation [11] or He initialisation [12]; the appropriate choice depends on the type of layer and the activation function used.

2.2.2 Forward Pass

The forward pass is the process by which input data flows through the network, layer by layer, to produce a prediction. At each layer, the inputs are multiplied by the weight matrix, summed with a bias term, and passed through a non-linear activation function. The output predictions are used to compute the loss, which drives the subsequent backward pass to update the model's parameters.

Data preparation Before entering the network, raw input data is prepared to be compute-ready as already described in Section 2.1.2.

Layer computation During the forward pass, each layer performs a series of computations on the input data, transforming it into a higher-level representation. These transformations are computed by means of matrix multiplications (MatMul) and activation functions.

Weighted sum and pre-activation Before the non-linear activation function is applied, the layer must compute the raw interaction between the input data and the model's learned parameters. This is achieved through a linear transformation, mathematically expressed as

$$z = Wx + b \tag{2.1}$$

where W represents the weight matrix, x is the input vector (or the output from the previous layer), and b is the bias vector. This intermediate result, known as the pre-activation, aggregates the learned feature representations before the activation function dictates each neuron's final output.

Activation functions In order for the neural networks to learn complex relationships in data, activation functions play a crucial role by introducing non-linearity in the computation. Common activation functions include sigmoid, tanh, GeLU [13], and ReLU [14].

2.2.3 Loss Function

The loss function, also referred to as cost or objective function, quantifies the disparity between the model's prediction and the ground truth. The loss provides a

2. Background

scalar signal that the training algorithm seeks to minimise, thereby enhancing the model’s predictive accuracy. A well-chosen loss function should be differentiable, reflect the true objective of the task, and behave well numerically across the expected range of outputs.

Choosing the right loss function The choice of loss function depends on the nature of the task. In regression tasks, where the objective is to predict continuous values, common loss functions include Mean Squared Error (MSE) and Mean Absolute Error (MAE). On the other hand, in classification tasks, where the aim is to assign inputs to a specific category, cross-entropy losses such as Binary Cross-Entropy and Categorical Cross-Entropy are the common choices.

2.2.4 Backward Pass

The backward pass, most commonly termed backpropagation, is the fundamental algorithm used to calculate the gradient of the loss function with respect to each parameter in the neural network. While the forward pass computes the predictions, backpropagation quantifies how much each individual weight and bias contributed to the final error. The primary purpose of the backward pass is to enable the model to learn from its mistakes by adjusting its parameters in a direction that minimises the loss function.

Gradient computation Backpropagation leverages the *chain rule* of calculus to efficiently compute the partial derivatives of the error backward through the network’s layers, effectively propagating the error signal in reverse. This process generates the gradients: the mathematical foundation that allows neural networks to learn. More specifically, gradients are vectors indicating the direction and magnitude of change that should be applied to each parameter to minimise the loss. A positive gradient indicates that increasing the parameter’s value will increase the loss, while a negative gradient suggests the opposite.

2.2.5 Optimisation Algorithm

Once the gradients are computed via backpropagation, an optimisation algorithm is required to update the network’s parameters and minimise the loss function. The optimiser dictates the strategy and the magnitude of the weight adjustments at each training step, effectively navigating the complex, high-dimensional “loss landscape” to reach an optimal or near-optimal solution.

Stochastic Gradient Descent (SGD) Stochastic Gradient Descent represents a foundational optimisation algorithm. Unlike standard Gradient Descent, which calculates the gradient using the entire training dataset, SGD estimates the gradient using randomly selected samples or batches (mini-batches) from the dataset. This stochastic process introduces noise that helps the model escape local minima and saddle points, significantly accelerating the training process on large datasets while maintaining an acceptable path towards convergence.

2.2.6 Learning Rate

The learning rate is a critical hyperparameter that controls the step size taken by the optimisation algorithm during each parameter update. A learning rate that is too large can cause the optimiser to overshoot the minimum, leading to divergence, whereas a learning rate that is too small results in excessively slow convergence. Finding the optimal learning rate, often through scheduling or warmup techniques, is essential for a stable and efficient training process.

2.2.7 Epochs and Batches

Training a deep neural network requires iterating over the dataset multiple times. An epoch is defined as one complete pass of the entire training dataset through the model. Since processing the whole dataset at once is computationally prohibitive, the data is partitioned into smaller, manageable subsets called batches. The network performs a forward pass, loss calculation, and weight update for each batch, enabling efficient memory utilisation and providing regular stochastic gradient estimates throughout the epoch.

Tuning the batch size allows practitioners to navigate the fundamental trade-off between hardware efficiency and optimisation quality. Larger batches accelerate the training process by maximising parallel computation and requiring fewer parameter updates per epoch. In contrast, smaller batches ensure more frequent weight adjustments, assuring a more robust convergence trajectory. The final choice relies heavily on hardware capacity, dataset dimensionality, and the inherent properties of the employed optimisation algorithm.

2.2.8 Validation and Evaluation

To ensure that a model is actually learning the underlying patterns of the data rather than simply memorising the training examples (*overfitting*), its performance must be rigorously evaluated on unseen data throughout and after the training process.

Validation set The validation set is a strictly disjoint subset of the data used during the training phase to monitor the model’s generalisation capabilities and predictive accuracy. It provides an unbiased evaluation of a model fit on the training dataset, acting as a proxy for real-world performance. By tracking the validation loss, practitioners can adjust the hyperparameters (e.g., learning rate, regularisation strength) and prevent overfitting by applying techniques such as early stopping or model checkpointing, ensuring the selection of the most robust and generalisable model.

Test set The test set is a holdout dataset that remains completely untouched during the training and hyperparameter tuning phases. It is utilised exclusively at the conclusion of the development pipeline to provide a definitive benchmark of the model’s generalisation capabilities. To ensure a reliable estimate of true real-world

2. Background

performance, the test dataset must be assembled without any selection bias, serving as an impartial and representative evaluator of the final model.

2.2.9 Hyperparameter Tuning

Unlike model parameters (weights and biases), which are learned automatically during training, hyperparameters are the external configuration variables set prior to the training process. Examples include the learning rate, batch size, number of layers, and the choice of the optimiser. Hyperparameter tuning is the systematic process of searching for the optimal combination of these settings in order to maximise the model’s predictive accuracy and convergence stability. As previously discussed, this process relies on a validation dataset to evaluate different configurations during training, ensuring the model’s capacity to generalise is measured without compromising the final test set.

2.2.10 Training, Fine-Tuning and Inference

The operational lifecycle of a deep learning model spans three distinct phases. Training is the resource-intensive process of initialising a model from scratch and iteratively adjusting its weights on a large dataset to learn general representations. Fine-tuning (or transfer learning) takes an already pre-trained model and adapts it to a narrower, domain-specific task using a smaller dataset, requiring significantly less computational effort while achieving high accuracy. Finally, inference is the deployment phase where the frozen, fully trained model is used to make predictions on new data, executing only the forward pass to guarantee low latency.

2.3 Federated Learning

Artificial Intelligence (AI) models require massive volumes of data typically centralised in a single location for model training. This opens up opportunities for any personally identifiable information (PII) contained in the datasets to be exposed during transmission or storage. Furthermore, transferring terabytes of data from distributed devices to a central server consumes enormous network bandwidth, incurs high storage costs, and introduces significant latency.

Federated Learning (FL) addresses these problems by training a shared model across distributed data holders without centralising their raw data [2]. FL allows the training process to be performed locally on each participant of the federation; consequently, only model updates are communicated to the central server, or aggregator. This paradigm shift in training allows for secure collaborative learning on sensitive data while reducing communication costs and ensuring strict regulatory compliance.

2.3.1 FL Setting

The federated learning setting typically involves the following steps:

1. **Model initialisation:** the central server initialises the global model, either randomly or from a previously saved checkpoint.
2. **Distribution and local training:** the model is then distributed among the participating edge devices or servers. Each client performs local training on its proprietary data for a predefined number of *local epochs*.
3. **Updates and aggregation:** at the end of the local training, each client node transmits back its model updates, which can either be the full model parameters or just the gradients. The central server aggregates these contributions to update the global model, which is subsequently redistributed to the clients.

Steps two and three constitute a so-called *round*, as illustrated in Figure 2.1. The training process iterates through multiple such rounds, terminating when convergence is achieved or a predefined stopping criterion is met (e.g., a maximum number of rounds).

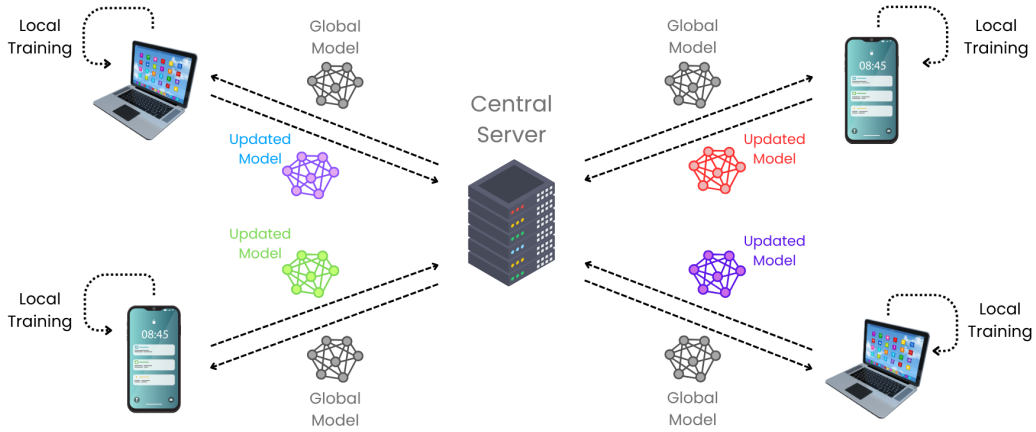


Figure 2.1: Round in a Federated Learning setting.

2.3.2 Advantages and Challenges

The primary advantage of FL is enhanced data privacy, as raw information never leaves the local node. This decentralised approach naturally mitigates the risks associated with centralised data breaches and facilitates compliance with strict privacy regulations (e.g., GDPR in Europe). Furthermore, it reduces communication overhead since only parameter updates, which are significantly smaller than raw datasets, are transmitted over the network. Finally, FL leverages the distributed computational resources of edge devices, effectively reducing the processing and storage burden on central servers.

2. Background

Despite its promising benefits, FL comes with several distinct challenges. Communication bottlenecks can arise, particularly over slow or unstable networks, as models often require thousands of communication rounds to converge. Additionally, the federation is susceptible to security threats such as model poisoning or inference attacks, where malicious actors attempt to compromise the global model or extract sensitive information from the shared gradients. Lastly, managing the inherent heterogeneity of the participating nodes and ensuring fault tolerance remain critical hurdles for scalable deployments.

2.3.3 Cross-Silo and Cross-Device

FL environments are broadly categorised into two settings based on the scale and reliability of the participants: *cross-silo* and *cross-device*.

- **Cross-silo** involves a relatively small number of reliable, high-capability clients, such as distinct organisations, hospitals, or financial institutions participating in collaborative training. These nodes typically possess high computational capabilities, large datasets, and stable connections.
- **Cross-device** scales to potentially millions of mobile or IoT (Internet of Things) edge devices. These clients are highly unreliable, constrained by battery and processing power, and exhibit intermittent availability, requiring formalised aggregation protocols to handle frequent dropouts and device unavailability.

2.3.4 Heterogeneity in Federated Learning

Real-world edge FL deployments exhibit heterogeneity across multiple dimensions at once. User devices collect different data, expose different processor and memory budgets, and return updates through networks with variable latency, bandwidth, and availability. Treating these diverse factors as a single, generic “heterogeneity problem” is ineffective. Instead, it is crucial to address this challenge across its distinct dimensions, as they severely impact both the convergence rate and the statistical efficiency of the global model.

Data heterogeneity (Non-IID) Local datasets are rarely *independent and identically distributed (IID)*. Each client gathers data based on its specific environment and usage patterns, leading to significant variations in data distribution, class representation, and dataset sizes across the federation. This statistical heterogeneity causes local models to diverge toward local optima, hindering the convergence of the global model and potentially degrading its generalisation capabilities.

Compute heterogeneity The participating devices in a federation often range from high-end servers to resource-constrained IoT sensors. This results in significant discrepancies in computational capabilities, memory capacity, and network bandwidth. Consequently, local training times can vary by orders of magnitude,

making synchronous aggregation strategies highly inefficient if the central server is forced to wait for the slowest clients.

Straggler-induced heterogeneity Directly stemming from compute heterogeneity and network instability, *stragglers* are clients that fail to complete their local training or transmit updates within a designated time window. In synchronous FL algorithms, a few stragglers can stall the entire training process. Handling this form of heterogeneity requires robust system designs, such as asynchronous aggregation, client sampling techniques, or dynamic deadline adjustments, to ensure continuous progress despite node delays or failures.

2.4 Mixture of Experts Architecture

This section provides an overview of the Mixture-of-Experts (MoE) architecture. Specifically, it introduces the core concept of the paradigm, the foundational limitations it addresses, and the internal routing mechanisms that enable its sparse execution.

2.4.1 Scaling Laws and Model Capacity

The evolution of deep learning has been largely driven by empirical *scaling laws* [1], which demonstrate that increasing a neural network’s parameter count consistently improves its representational capacity and generalisation capabilities.

Driven by this dynamic, modern state-of-the-art architectures feature increasingly massive parameter counts. Contemporary models exceed hundreds of billions of parameters, with frontier architectures reaching the one-trillion parameter mark.

The dense bottleneck However, traditional neural networks rely on a *dense, fully-connected* architectural paradigm. In a dense model, every single parameter must be activated and computed during every forward and backward pass, regardless of the specific input being processed. This monolithic approach creates a severe scaling bottleneck: as models grow in size, the computational cost, memory bandwidth requirements, and inference latency scale proportionally.

Training and deploying such extremely large dense models quickly become physically and economically prohibitive, exceeding the practical limits of standard hardware accelerators and making them entirely unsuitable for resource-constrained environments like edge devices.

2.4.2 Sparsity and Conditional Computation

Modern deep learning models have been demonstrated to be highly over-parameterised, meaning that a significant portion of their learned representations is actually stored in a small portion of the model. This phenomenon led to a wide research field focused on the *sparsification* and *conditional computation* of such architectures to improve their efficiency.

2. Background

Sparsity At its core, sparsity is the structural opposite of dense computation. While traditional neural networks require every single weight matrix to interact with every input tensor, a sparse architecture dynamically zeroes out irrelevant neural pathways based on the specific data being processed. Sparsity enables the creation of highly over-parameterised models that retain the computational footprint of significantly smaller networks.

Conditional computation The principle of conditional computation dictates that a deep neural network should possess the algorithmic ability to dynamically determine which of its internal sub-components to activate based purely on the specific input it receives (e.g., a vision model should not waste computational cycles activating components tuned for analysing textual characters if the network is currently processing an image of a natural landscape).

2.4.3 MoE Formulation

The Mixture of Experts (MoE) architecture, originally formalised in 1991 by Jacobs et al. [3] and extended by Shazeer et al. [4], fundamentally resolves the previously examined limitations by shifting from a dense to a sparse paradigm. Rooted in the concept of conditional computation, MoE systematically decouples a model’s total parameter capacity from its active computational cost by activating only a sparse subset of parameters for each input.

The experts Instead of forcing every input to traverse the entire network, MoE replaces massive dense layers with an ensemble of smaller, highly specialised sub-networks, known as *experts*. A dynamic *routing mechanism* evaluates each input and activates only a minute fraction of these experts for processing. Consequently, an MoE model can maintain a massive total parameter count, storing vast amounts of learned knowledge, while strictly requiring the active computational resources (FLOPs) of a significantly smaller model during inference.

The router To dynamically select which experts to activate based on the incoming input, the architecture employs a *router* (also referred to as a gating network). The router is a learnable neural network that analyses each input and routes it to the most capable experts. For a given input representation $x \in \mathbb{R}^d$, the router first computes raw scores (logits) by projecting x onto a trainable weight matrix $W_g \in \mathbb{R}^{d \times N}$, where N is the total number of experts. The scores assigned to each expert represent the “confidence” or the “fitness” assigned by the router for that specific input.

Top-k selection However, to enforce sparsity and achieve the benefits of conditional computation, a *Top-k* selection mechanism is applied. The router identifies the k experts with the highest raw scores (where $k \ll N$, typically $k = 1$ or $k = 2$) and explicitly forces the logits of the remaining $N - k$ experts to $-\infty$. Finally, a *Softmax* function is applied to the routed scores to normalise them into a discrete probability distribution in the range $[0, 1]$. This ensures that the gating weights of

the selected experts are strictly positive and sum exactly to one. The final output of the MoE layer is thus computed as:

$$y = \sum_{i \in \mathcal{E}_s(x)} G(x)_i E_i(x) \quad (2.2)$$

where $\mathcal{E}_s(x)$ denotes the set of indices of the selected top- k experts for input x , $G(x)_i$ represents the normalised routing probability assigned to the i -th expert, and $E_i(x)$ is the output of that specific expert.

A visual representation of this sparse gating mechanism is shown in Figure 2.2.

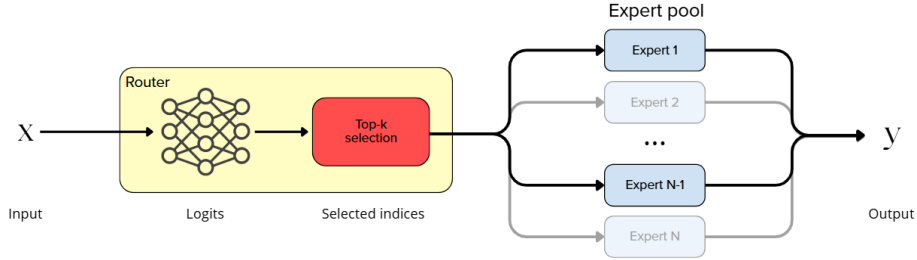


Figure 2.2: Routing mechanism in MoE. Top-2 selection.

2.4.4 Routing Challenges and Load Balancing

While the sparse routing mechanism provides significant efficiency gains, it introduces unique optimisation challenges during training.

Expert collapse Without proper constraints, the router network is prone to a self-reinforcing failure known as *expert collapse* or *routing collapse*. The network might rapidly learn to favour a small subset of experts, or even just one, continuously routing the majority of data to them. Since these experts are exposed to more information, their representations become highly refined, causing the router to reward them with even more inputs. Consequently, the remaining experts become “dead” or underutilised, defeating the purpose of the architecture and creating severe memory and computational bottlenecks on the hardware hosting the overloaded experts.

Auxiliary loss To prevent this phenomenon and enforce a balanced utilisation of the expert pool, an *auxiliary load-balancing loss* is commonly added to the primary training objective. This mathematical regularisation term penalises the model if the distribution of routed tokens heavily skews toward a few experts, ensuring that all sub-networks receive a comparable amount of training signal. In order to be effective without dominating the main training objective, the auxiliary loss is scaled by a weighting hyperparameter α (typically set to small values such as 0.01 or 0.001). The overall loss function optimised by the network can be formulated as:

$$\mathcal{L} = \mathcal{L}_{task} + \alpha \mathcal{L}_{aux} \quad (2.3)$$

2. Background

where $\mathcal{L}_{task}$ is the task-specific loss function (e.g., Cross-Entropy Loss for classification or Mean Squared Error for regression), and $\mathcal{L}_{aux}$ represents the load-balancing penalty.

2.4.5 Architectural Trade-offs

The adoption of the Mixture of Experts paradigm represents a profound engineering trade-off between computational efficiency and infrastructural demands. By decoupling representational capacity from active compute, MoE enables the pre-training of massive models that naturally specialise in diverse syntactic patterns and knowledge domains. This sparse execution yields faster convergence and superior performance compared to dense networks with equivalent inference costs. However, this theoretical efficiency imposes severe hardware constraints. While only a fraction of the network is computationally active during a forward pass, the complete set of expert parameters must reside in the GPU VRAM to prevent severe memory-fetching latency. Furthermore, scaling these models requires complex distributed systems, where routing tokens to the correct hardware nodes introduces substantial communication overheads through a mechanism known as *Expert Parallelism*.

2.5 Vision Transformers

This section presents the Vision Transformer (ViT) architecture, originally introduced by Dosovitskiy et al. [6]. By leveraging a pure transformer architecture inherited from Natural Language Processing (NLP), ViTs represent a fundamental paradigm shift in computer vision. They actively challenge the long-standing dominance of Convolutional Neural Networks (CNNs), achieving state-of-the-art performance on image recognition tasks when pre-trained at scale.

2.5.1 From Convolutions to Patch Sequences

Traditional CNNs are built upon strong inductive biases, specifically translation equivariance and spatial locality, which allow them to efficiently extract features from pixel grids. ViT discards these architectural priors, treating images as one-dimensional sequences of *patches*. In this paradigm, image patches are treated exactly the same way as tokens (words) in an NLP application. Following this analogy, and for consistency with the Transformer and Large Language Model literature, throughout this thesis the terms *patch* and *token* are used interchangeably to denote the elementary input units processed by the model.

2.5.2 Architecture Overview

In their foundational work, the authors explicitly designed the ViT to adhere as closely as possible to the original Transformer architecture (Vaswani et al. [15]). The primary advantage of this setup (illustrated in Figure 2.3) is that highly scalable NLP Transformer architectures can be leveraged almost out of the box for computer vision tasks.

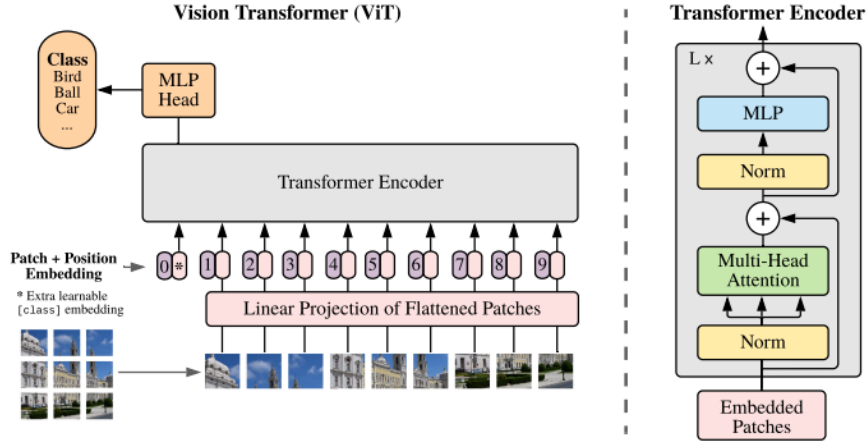


Figure 2.3: Vision Transformer architecture. Source: [6]

Patch extraction To process 2D images using a standard transformer, the original image $x \in \mathbb{R}^{H \times W \times C}$ is reshaped into a 1D sequence of flattened patches $x_p \in \mathbb{R}^{N \times (P^2 \times C)}$, where (H, W) is the image resolution, C is the number of channels (e.g., RGB colours), and (P, P) is the resolution of each individual patch, defining the *patch size*. Typical patch sizes are 16×16 or 32×32 . The resulting number of patches is $N = HW/P^2$, which serves as the effective *input sequence length* for the transformer.

Patch embedding Raw flattened patches cannot be fed directly into the network, since transformers inherently process sequences of vectors with a constant latent dimension $\mathcal{D}$. Thus, they are mapped to $\mathcal{D}$ dimensions using a *trainable linear projection*, practically realised through a 2D convolution. The output of this projection constitutes the *patch embeddings*, effectively serving the exact same purpose as word embeddings in NLP applications.

Classification token Inspired by the BERT [16] architecture, an additional learnable embedding, known as the [CLS] (*classification*) token, is prefixed to the sequence of embedded patches. The state of this specific token serves as the aggregated, global representation of the entire image. It will eventually be processed by the classification head to generate the final prediction, as detailed subsequently.

Positional embedding Because the self-attention mechanism is inherently permutation-invariant, a pure transformer contains no built-in notion of sequence order or spatial arrangement. To retain positional information, standard *learnable 1D position embeddings* are mathematically added to the patch embeddings before they enter the encoder. This is crucial for re-injecting spatial awareness, allowing the model to infer the original arrangement of the flattened patches.

Transformer encoder The core sequence (composed of the patch embeddings, the [CLS] token, and the added positional information) is processed by a stack of

2. Background

identical *Encoder* blocks. Following the standard transformer design, each block consists of two primary sub-layers: a Multi-Head Self-Attention (MSA) mechanism and a Multi-Layer Perceptron (MLP), also referred to as a Feed-Forward Network (FFN). Standard Layer Normalisation (LN) is applied *before* every sub-layer, and *residual connections* are employed after every sub-layer to ensure stable gradient flow during deep training.

Classification head To produce the final prediction, a *classification head* is attached exclusively to the final state of the [CLS] token. For standard image classification tasks, this component is typically implemented as a basic MLP or a single linear projection mapping the $\mathcal{D}$ -dimensional representation to the required number of target classes. Crucially, this head is strictly task-dependent; when adapting the Vision Transformer for other computer vision objectives (e.g., depth estimation or semantic segmentation), the classification head is discarded and replaced with appropriate task-specific modules.

2.5.3 Scaling and Performance

The removal of image-specific inductive biases makes ViT highly reliant on massive amounts of data to learn spatial relations from scratch.

Data requirements When trained on mid-sized datasets such as ImageNet without strong regularisation, ViT models yield modest accuracies, typically underperforming compared to equivalent ResNets. However, the authors demonstrated that this behaviour reverses when the models are pre-trained on enormous datasets, such as ImageNet-21k or JFT-300M.

State-of-the-art results Under intensive pre-training regimes, ViT successfully overtakes CNNs, matching or exceeding state-of-the-art convolutional baselines. These results demonstrate that large-scale training successfully overcomes inductive bias, solidifying the Vision Transformer as a scalable and competitive architecture in modern computer vision.

2.6 Vision Mixture-of-Experts

As an extension of the previously detailed ViT, Riquelme et al. [5] successfully integrated the sparsely gated Mixture-of-Experts paradigm into computer vision, introducing the *Vision Mixture-of-Experts* (V-MoE).

This architecture is of particular relevance, as it forms the basis of the model employed in this thesis.

2.6.1 The MoE Block and Expert Architecture

The V-MoE architecture fundamentally replaces a subset of standard encoder blocks with *MoE blocks*. The only difference between them lies in the feed-forward

network, where the dense MLP is substituted with a *sparse MoE layer*. The number and position of MoE blocks typically follow one of two design patterns: they can be positioned at every odd-numbered block (as illustrated in Figure 2.4), or restricted to the *last-k* odd blocks (e.g., on the last-2 odd blocks).

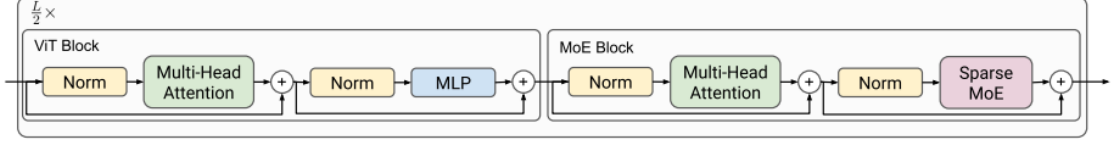


Figure 2.4: V-MoE structure. Source: [5]

Sparse MoE layer At the core of this layer lies a gating network, which dynamically evaluates each input patch and assigns it to the top- k most suitable experts. In this way, routing is conducted per-patch (or per-token) and not per-image. Consequently, different regions of the same image (e.g., a background sky versus a complex foreground object) may be processed by entirely different sets of experts.

Expert overview In the original formulation of the paper, every individual expert is implemented as a Multi-Layer Perceptron (MLP), mirroring the ViT baseline. Internally, this MLP processes the patch through a sequence of three operations (Figure 2.5):

1. An *up-projection* linear layer expands the vector’s latent dimension by a factor of four ($4 \times \mathcal{D}$).
2. A non-linear *Gaussian Error Linear Unit* (GeLU) activation function is applied to this expanded representation.
3. A *down-projection* linear layer compresses the vector back to the model’s baseline latent dimension $\mathcal{D}$.

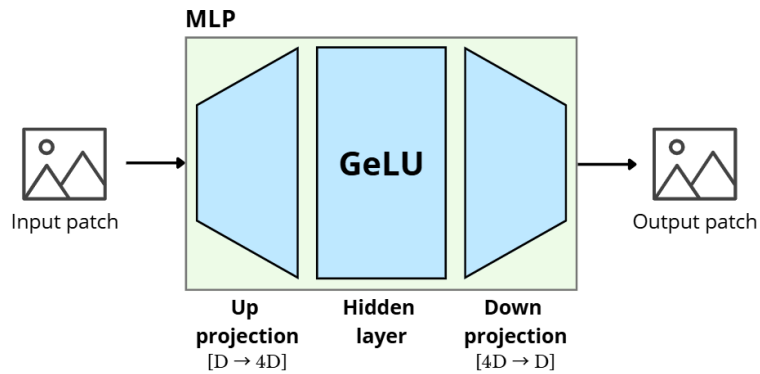


Figure 2.5: MLP structure

2. Background

2.6.2 Expert Capacity and Token Dropping

One of the most critical challenges in deploying MoE architectures on standard hardware (such as TPUs or GPUs) is memory management. Hardware accelerators require static tensor shapes, meaning that the exact number of patches processed by each expert must be defined before the computation begins.

Buffer capacity To enforce static shapes, each expert is assigned a fixed *buffer capacity*, defined as the maximum number of tokens it can process in a single batch. The buffer capacity B is mathematically determined by the total number of tokens in the batch N_{total} , the number of experts E , the routing factor k , and a hyperparameter known as the *capacity factor* C (typically set to 1.25 or 1.0):

$$B = \text{round} \left(\frac{N_{total} \cdot k}{E} \cdot C \right) \quad (2.4)$$

Token dropping Consequently, a heavily routed expert can only process a finite number of tokens, forcing any excess patches beyond its capacity B to be *dropped*. On the other hand, under-utilised experts must pad their remaining buffer space with dummy zero-tokens to satisfy the rigid size constraints. Crucially, a dropped token is not entirely discarded from the network; rather, it receives zero contribution from that specific MoE layer, bypassing the expert entirely and propagating unaltered to the subsequent block via the standard residual connection.

2.6.3 Scaling Efficiency and Performance

The primary motivation behind the introduction of the MoE paradigm in computer vision is to overcome the scaling bottlenecks of dense architectures. In a standard ViT, scaling up the model to capture more complex representations inevitably leads to a proportional, often prohibitive, increase in computational cost (FLOPs). By decoupling the parameter count from the computational requirements, V-MoE enables a highly efficient scaling trajectory.

Compute-matched comparisons When evaluating models with an equivalent inference cost (e.g., equal active parameters or FLOPs), the sparse architecture significantly outperforms its dense counterpart. Riquelme et al. [5] demonstrated that, given an equivalent number of inference FLOPs, V-MoE models achieve a gain of approximately 1% in ImageNet few-shot (5-shot) accuracy and nearly 3% in the JFT precision@1 metric. During inference, a properly tuned V-MoE can match the state-of-the-art accuracy of massive dense models using less than 50% of the computational operations and operating in under 60% of the runtime.

Parameter-matched comparisons and memory footprint Conversely, comparing a V-MoE to a dense ViT with the exact same number of *total* parameters requires a precise distinction between *static memory* and *computational load*. Since the sparse model must still allocate all expert weights on the hardware regardless

of whether they are activated, V-MoE does not provide savings in static memory footprint. The true efficiency breakthrough lies in the *runtime computation*: by applying only a small subset of parameters (e.g., $k = 1$ or $k = 2$ experts) to each individual patch, the V-MoE drastically reduces the memory allocated for runtime activations and the overall FLOPs, processing images significantly faster without degrading top-1 accuracy.

Chapter 3

Related Work

Contents

3.1	Partial Expert Replication in Distributed MoE Training	27
3.1.1	Problem Statement	27
3.1.2	The Partial-Replica Paradigm	28
3.1.3	Expert Assignment Strategies	30
3.1.4	Evaluation and Results	30
3.1.5	Relevance and Implications for the Thesis	31
3.2	Parameter-Efficient Fine-Tuning	32
3.2.1	Low-Rank Decomposition	32
3.2.2	Architectural Advantages	32
3.2.3	Relevance and Implications for the Thesis	33
3.3	Sparsity-aware Aggregation in FL	33
3.3.1	FedAvg and its Limitations	34
3.3.2	Sparsity-aware Model Aggregation (SaMA)	34
3.3.3	Relevance and Implications for the Thesis	35

This chapter details the three foundational lines of work upon which this thesis is built. Each addresses a specific bottleneck in fine-tuning a Mixture-of-Experts (MoE) model across resource-constrained, federated edge devices:

- **The cost of *distributing* an MoE model:** Standard distributed training requires every node to replicate the full model. We examine how partial expert replication, combined with a *skip-token* mechanism, breaks this barrier and forms the methodological core of our approach.
- **The cost of *adapting* a large pre-trained model:** Full fine-tuning is prohibitive for edge hardware and constrained networks. We review a Parameter-Efficient Fine-Tuning strategy that trains only a small, low-rank update while freezing the backbone.
- **The cost of *aggregating* sparse updates:** Standard FedAvg dilutes experts by averaging them uniformly across clients, even those that never used them. We analyse a sparsity-aware aggregation scheme that weights each expert by its actual token utilisation.

Taken together, these three pillars provide the essential building blocks to transfer MoE training from large-scale datacentre environments to the federated fine-tuning of a Vision MoE on heterogeneous edge devices.

3.1 Partial Expert Replication in Distributed MoE Training

Mixture-of-Experts architectures reduce computational cost through sparsity, but not communication: when training is distributed, every node must still replicate the full expert pool and exchange the entire model at each step. This *full-replica barrier* ties memory and bandwidth to the global model size. In this section, we explore *FoMoE*, the framework introduced by Sani et al. [7] that breaks this barrier. It serves as the core baseline for this thesis, providing the methodological foundation that we adapt to a novel context.

3.1.1 Problem Statement

Pre-training Large Language Models (LLMs) conventionally requires large-scale infrastructures equipped with tightly coupled hardware accelerators.

Mixture-of-Experts architectures have recently achieved state-of-the-art results by decoupling parameter count from computational cost. This efficiency enables training massive models on constrained compute budgets, yet they still require said compute to be co-located in large datacentres with *high-bandwidth interconnects* between compute nodes.

3. Related Work

Towards distributed training: Photon and DiLoCo To overcome these physical constraints, recent approaches such as Photon [17] and DiLoCo [18] utilise low-communication methods to enable scaling across geographically distributed, weakly connected datacentres over Wide Area Networks (WANs). These algorithms successfully reduce the bandwidth overhead by limiting the synchronisation frequency during the training lifecycle.

Specifically, these methods are based on the principle of *Local SGD*. Instead of synchronising gradients across nodes after every single training step, which would immediately bottleneck over slow WAN links, each distributed worker performs multiple local optimisation steps independently. Only after a predefined number of local steps, the workers communicate to aggregate their updates, often relying on an *outer optimiser* at the server level to maintain global convergence.

Full-replica barrier However, a critical inefficiency remains: both these methods still treat the model as a *dense, monolithic* entity. This paradigm forces every participating site to store the entire model state and exchange full-parameter payloads at every synchronisation step. This systemic limitation constitutes the so-called *full-replica barrier*.

3.1.2 The Partial-Replica Paradigm

To enable the training of frontier models across geographically distributed datacentres that lack the dedicated high-performance interconnects and memory availability of premium ones, FoMoE fundamentally breaks the full-replica paradigm by co-designing the training strategy with the structural sparsity of MoE models. Rather than maintaining identical monolithic replicas at every site, FoMoE *partitions each expert layer*, assigning specific subsets of expert modules to different distributed workers.

Model decomposition Conceptually, FoMoE categorises the model parameters θ into two distinct subsets:

- **Dense components** (θ_{dense}): Fully replicated parameters, including attention weights, layer norms, shared experts, and input/output embeddings.
- **Sparse components** (θ_{sparse}): Potentially partitioned parameters, specifically the expert weights (and, in the standard configuration, their routing networks).

While scaling depth adds dense parameters, scaling width adds experts (sparse parameters). Consequently, as the overall model scale grows, the ratio θ_{sparse}/θ shifts significantly. This relationship demonstrates that optimising the synchronisation of these sparse components is the primary lever for enhancing communication efficiency in large-scale distributed training.

Experts partitioning Each expert is allocated among the M workers according to the *overlapping factor* $O_e \in \{1, \dots, M\}$, defined as the number of distinct workers hosting any specific expert. When $O_e = 1$, each expert is strictly assigned to a single worker, while $O_e = M$ ensures every worker maintains the entire pool of experts, effectively reverting to a full-replica scenario. By partitioning the experts such that $O_e < M$, the per-worker memory footprint for expert weights is reduced by a factor of M/O_e .

The major consequence of this overlapping factor is that each worker hosts only a subset of the expert pool, rather than the entire collection. This is how the partial-replica regime is achieved: sparse parameters (experts) are partially distributed, while the dense backbone is fully shared across all workers.

Ghost experts Building on the partial-replica strategy, FoMoE introduces an alternative routing configuration to further optimise computational efficiency. Rather than partitioning the routing networks alongside the experts, the routing matrices are maintained in full across all workers, effectively treating them as part of the dense parameters (θ_{dense}). This approach adds minimal communication overhead, since the routers' parameter count is negligible compared to the expert modules.

Thus, a local router is free to assign tokens to experts that are not locally hosted by that worker. These remote, locally unavailable expert modules are referred to as *Ghost Experts*.

Skip tokens When a token is assigned to a ghost expert, the local worker lacks the corresponding weights to execute the feed-forward network computation. Instead of transferring tokens across the Wide Area Network (WAN) to perform remote computation (as seen in high-bandwidth dispatch mechanisms like DeepSeek's), FoMoE introduces the *skip-token* mechanism. As illustrated on the right-hand side of Figure 3.1, the computational graph dynamically bypasses the execution for the missing ghost expert. Essentially, this mechanism trades a slight decrease in active parameters per token for zero cross-node activation transfers. This is a crucial benefit, as these transfers would easily overwhelm low-bandwidth links.

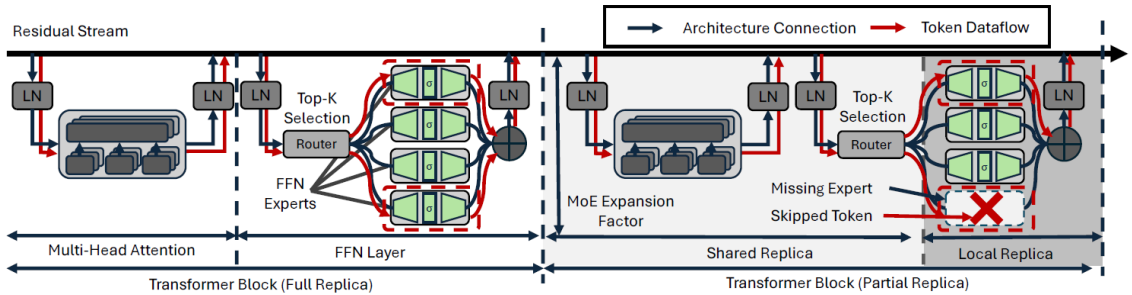


Figure 3.1: Comparison between a transformer MoE block as a full versus partial replica. Source [7]

3. Related Work

3.1.3 Expert Assignment Strategies

Once the expert partitioning configuration is established, the system must dictate how these sparse components are mapped and distributed across the available computing nodes. To navigate the trade-offs between communication efficiency and model generalisation, the framework analyses two distinct assignment modalities: *fixed* and *random* placement.

Fixed placement Under this placement mode, the central server deterministically maps each expert to specific workers at the beginning of the training process. These assignments remain strictly invariant throughout the entire training lifecycle, meaning that each participating site is responsible solely for updating its predefined local subset of experts. The primary advantage of this strategy is communication efficiency: by pinning experts to specific nodes, the system enforces strict data locality and entirely eliminates the need to migrate expert weights across the network during the training process.

Random placement In contrast, the random placement strategy introduces a periodic reshuffling mechanism. At a predefined cadence of $\mathcal{T}$ synchronisation rounds, the central server randomly redistributes the experts across the workers, ensuring that each expert dynamically migrates through different network nodes over time. The crucial benefit of this dynamic reassignment is that it acts as a powerful regulariser. By continuously decoupling experts from specific local data shards, the routing networks and the experts themselves are exposed to the full diversity of the global data distribution, which significantly enhances generalisation. However, this strategy inherently introduces a communication penalty, as migrating experts requires transmitting their weights across the network.

Trade-offs and use cases The choice between fixed and random assignment fundamentally depends on the hardware and network profile of the distributed environment. Fixed assignment is highly optimised for extremely bandwidth-constrained settings, where minimising the communication payload is the absolute priority to make training feasible. Conversely, random assignment generally outperforms its fixed counterpart in terms of final machine learning quality. Therefore, random assignment is the preferred scenario when the network can tolerate the amortised cost of periodic state migration in exchange for superior model generalisation and robustness against highly heterogeneous local data distributions.

3.1.4 Evaluation and Results

The authors comprehensively evaluated the FoMoE framework across various model scales, ranging from small models (150M parameters) up to massive (100B) parameter configurations.

Architecture and Expert Configuration Specifically, the empirical evaluations were conducted using decoder-only Large Language Models (LLMs) based on

standard Transformer blocks. Models from 150M up to 100B parameters were configured with a total pool of 128 experts ($N_e = 128$). The number of active experts per token varied depending on the scale, ranging from $k = 4$ for the 150M model to $k = 32$ for the 100B model.

Communication and wall-clock efficiency The empirical results demonstrate that FoMoE establishes a new, superior Pareto frontier for cross-datacentre training. By transmitting only a fraction of the sparse parameters at each synchronisation step, the partial-replica system reduces overall communication costs by up to $1.42\times$ compared to efficient low-communication baselines, and by $45.44\times$ over standard DDP. Consequently, this dramatic reduction in payload size accelerates the total training wall-clock time across all tested WAN bandwidths (from 1 Gbps to 1000 Gbps). The authors note that these speedups are especially critical in severely bandwidth-constrained environments (e.g., 1 Gbps), where minimising the overlapping factor O_e is essential to make training feasible.

Computational throughput and the skip-token impact The evaluation confirms the theoretical benefits of the unpartitioned routing and skip-token mechanisms. By dynamically bypassing the feed-forward computation for ghost experts, FoMoE achieves an empirical throughput speedup of up to $1.4\times$. Notably, the study proves that this dynamic dropping of tokens does not negatively affect training convergence or final perplexity. Instead, it allows the system to achieve high model quality with significantly less compute. Furthermore, the authors highlight that these computational savings (FLOPs reduction) scale linearly and become increasingly pronounced at larger model scales, where sparse parameters constitute a dominant portion of the architecture.

3.1.5 Relevance and Implications for the Thesis

The FoMoE framework plays a crucial role as the architectural cornerstone for this thesis. In particular, the core paradigm of *partial expert replication*, alongside the *skip-token* mechanism, represents the main methodological contribution extracted and integrated into our experimental design.

In contrast with the original work’s focus on pre-training LLM architectures across distributed, weakly connected datacentres, this thesis targets a different scenario. Our project examines the *fine-tuning* process of a Vision Mixture-of-Experts architecture conducted within a real-world cross-device Federated Learning environment. The latter comprises highly heterogeneous edge devices severely constrained by computational power, memory and network bandwidth.

The primary objective is therefore to investigate whether the significant benefits in communication reduction, computational throughput, and wall-clock efficiency demonstrated by FoMoE can be successfully transferred to computer vision architectures (based on Transformers and Attention mechanisms) within a real-world federated setting.

Consequently, certain aspects of the original FoMoE framework fall outside the scope of our work. For instance, rather than training the full sparse parameters

from scratch, we propose injecting these communication-efficient methodologies into Parameter-Efficient Fine-Tuning (PEFT) techniques. To achieve this, the next section will explore the second foundational pillar of our project: *Low-Rank Adaptation (LoRA)*.

3.2 Parameter-Efficient Fine-Tuning

As neural network architectures scale to billions of parameters, the traditional paradigm of full fine-tuning (which requires updating and storing all model parameters for every downstream task) has become prohibitively expensive. To address this computational bottleneck, Hu et al. [8] introduced Low-Rank Adaptation (LoRA), a highly influential Parameter-Efficient Fine-Tuning (PEFT) technique designed to adapt massive pre-trained models with minimal resource requirements.

3.2.1 Low-Rank Decomposition

The foundational hypothesis of LoRA is that the weight updates required for task-specific adaptation reside on a low “intrinsic rank” subspace. Thus, instead of updating the fully dense pre-trained weight matrices, LoRA completely *freezes* the original weights (W_0) and injects two trainable rank decomposition matrices into the architecture.

Mathematically, the fine-tuning process through LoRA aims to produce an updated weight matrix W by adding a task-specific update ΔW to the original pre-trained state, such that $W = W_0 + \Delta W$. For a given pre-trained matrix $W_0 \in \mathbb{R}^{d \times k}$, LoRA approximates this update ΔW via a low-rank decomposition:

$$\Delta W = BA$$

where $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times k}$. The rank r is strictly constrained to be much smaller than the full dimensions ($r \ll \min(d, k)$). Consequently, the modified forward pass for an input vector x (as illustrated in Figure 3.2) is computed as:

$$h = Wx = (W_0 + \Delta W)x = W_0x + BAx$$

Crucially, during the optimisation phase, only the low-rank matrices A and B receive gradient updates, while the pre-trained state W_0 remains entirely unchanged.

3.2.2 Architectural Advantages

Compared to other PEFT techniques, such as standard *adapter layers* [19] or *prefix-tuning* [20], LoRA introduces critical structural improvements:

Massive reduction in resource costs By optimising only the low-rank matrices, LoRA reduces the number of trainable parameters by up to $10,000\times$ and lowers GPU memory requirements by up to $3\times$, all while matching or exceeding the performance of full fine-tuning across various benchmarks.

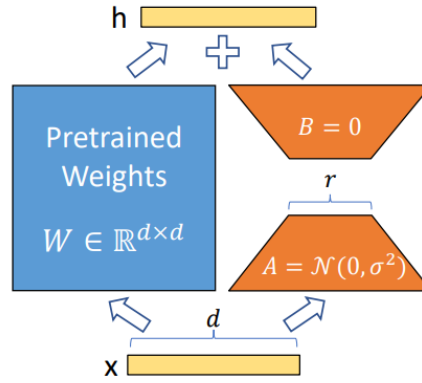


Figure 3.2: LoRA reparametrisation. Source: [8]

Zero additional inference latency Unlike traditional adapters that add sequential depth to the network, LoRA’s linear design allows the learned matrices to be explicitly merged with the frozen pre-trained weights prior to deployment ($W = W_0 + BA$). This paradigm guarantees no extra inference latency compared to a fully fine-tuned model.

3.2.3 Relevance and Implications for the Thesis

The adoption of LoRA is highly strategic for the context of this thesis: in a real-world, cross-device Federated Learning environment, edge devices are severely constrained by VRAM capacity and WANs cannot tolerate the transmission of entire model gradients.

Instead of performing a full fine-tuning of the V-MoE architecture, this project leverages LoRA to adapt them efficiently. By freezing the vast majority of the pre-trained base model and applying low-rank adaptations, the system drastically bounds both the local memory footprint required for optimiser states and the payload size transmitted over the network.

Ultimately, while FoMoE reduces the base architecture’s active parameters by distributing the experts, LoRA significantly minimises the dimensionality of the required training updates, making the federated fine-tuning process computationally and communicatively feasible on the constrained target setting.

3.3 Sparsity-aware Aggregation in FL

The third and final pillar of this work is built upon the *HFedMoE* framework introduced by Fang et al. [9]. While the original paper proposes a comprehensive suite of system-level optimisations for heterogeneous devices in Federated Learning (FL), our analysis focuses specifically on its novel *Sparsity-aware Model Aggregation* scheme, which resolves a critical bottleneck in federated MoE training.

3.3.1 FedAvg and its Limitations

In standard Federated Learning, the central server combines client updates using the *Federated Averaging (FedAvg)* algorithm, originally introduced by McMahan et al. [2]. As the foundational aggregation protocol, FedAvg dictates the exact rules by which the server merges independent local models into a single global state. Crucially, FedAvg operates on the principle that not all client contributions are statistically equal. In real-world federated networks, clients are inherently heterogeneous and hold varying amounts of local data. Treating every update equally could give disproportionate weight to low-data clients, potentially skewing the global model towards their specific data distributions. To counteract this, FedAvg enforces a *weighted aggregation* scheme: a client’s influence on the global model is strictly proportional to the number of training examples it has processed.

FedAvg formulation Formally, given a set of K participating clients, where each client k holds a local dataset of size n_k , the global model weights w at communication round $t + 1$ are updated as:

$$w_{t+1} = \sum_{k=1}^K \frac{n_k}{n} w_{t+1}^k \quad (3.1)$$

While highly effective for traditional, dense architectures, this paradigm breaks down when applied to Mixture-of-Experts architectures.

The challenge of sparse routing MoEs rely on sparse routing, meaning that inputs are directed only to a select subset of experts. In a federated setting, where a client’s local data is often highly heterogeneous, this routing mechanism naturally ends up favouring certain experts over others. As a result, the different experts within MoE layers may process vastly different amounts of data during the local training phase.

FedAvg determines the aggregation weights based on the total number of examples processed by the client, and applies that exact same weight uniformly to every parameter in the model. By doing so, it completely ignores the inherent sparsity of the architecture, blindly averaging an expert’s weights across all clients, even those that routed zero tokens to it.

3.3.2 Sparsity-aware Model Aggregation (SaMA)

To resolve this dilution effect, HFedMoE introduces a novel *Sparsity-aware Model Aggregation (SaMA)*. Instead of computing a global average weighted by the overall dataset size, this paradigm performs an independent aggregation for each expert, strictly based on its actual utilisation.

During the local training phase, each client tracks the exact number of tokens routed to every individual expert. At the synchronisation step (end of round), the central server aggregates the expert parameters by weighting each client’s contribution proportionally to the specific token count processed by that expert. Conse-

quently, if an expert on a given client processes no tokens, its weight contribution to the global aggregation is effectively zero.

This targeted mechanism ensures that the global expert parameters are dominated only by the updates coming from clients that actively utilised and trained them, preserving the integrity of the learned representations.

3.3.3 Relevance and Implications for the Thesis

In our target federated setting, the highly non-IID nature of local data across edge devices causes expert activations to be highly skewed. Furthermore, under the FoMoE partial-replica strategy discussed previously, clients only host and update a specific subset of the expert pool. Thus, several experts are physically absent from certain clients, meaning they receive no local updates whatsoever from the data available on those devices.

Applying SaMA at the server level is therefore essential for our architecture. By aggregating the LoRA update matrices based on explicit token routing counts rather than generic dataset sizes, our framework ensures that the specialised, low-rank adaptations are correctly aggregated, avoiding the severe parameter dilution that would otherwise compromise the fine-tuning process.

Chapter 4

Methodology: Partial-Replica V-MoE in Federated Learning

Contents

4.1	Objectives and Contributions	37
4.1.1	Partial-Replica Paradigm in FL	37
4.1.2	Static Expert Assignment	38
4.1.3	Dynamic Expert Reassignment and Trade-off Exploration	38
4.2	Architectural Analysis for Edge Deployment	39
4.3	Fine-Tuning in Edge Federated Learning	41
4.3.1	The PEFT Paradigm	41
4.4	Model Selection: V-MoE Small/32	41
4.4.1	Choice Rationale	42
4.4.2	Architectural Overview	44
4.5	Fine-Tuning Setting	45
4.6	Sparsity-aware Model Aggregation	46
4.6.1	Client-Side: Token Tracking Mechanism	46
4.6.2	Server-Side: Dual-Track Aggregation	47
4.7	Partial-Replica Setup	48
4.7.1	Static Assignment	48
4.7.2	Practical Deployment	49
4.8	Dynamic Expert Assignment	51
4.8.1	Methodology	51
4.8.2	Swap Rules	52
4.8.3	Swap Evaluation Metric	53
4.8.4	Reassignment Algorithm	55

This chapter presents the design and methodology of the federated learning system at the core of this thesis: a Vision Mixture-of-Experts (V-MoE) fine-tuned across resource-constrained, heterogeneous edge devices under a partial-replica regime. It first states the research objectives and structural contributions, then motivates the architectural choices required for edge deployment, introducing the selected V-MoE-S/32 backbone and the LoRA-based Parameter-Efficient Fine-Tuning (PEFT) strategy adopted.

Building on these foundations, the chapter details the implementation of the three methodological pillars of the proposed system: the Sparsity-aware Model Aggregation (SaMA) scheme, the static partial-replica assignment, and, finally, the dynamic expert reassignment protocol. Together, these components define an end-to-end methodology for investigating the accuracy, communication, and memory trade-offs of partial-replica V-MoEs in cross-device federated fine-tuning.

4.1 Objectives and Contributions

The primary objective of this thesis is to bridge the fundamental gap between the representational power of Vision Mixture-of-Experts (V-MoE) architectures and the severe hardware and bandwidth constraints inherent to real-world, cross-device Federated Learning (FL). As already detailed in Section 3.1, while FoMoE has successfully shattered the full-replica barrier, this methodology was exclusively tailored for the from-scratch pre-training of Large Language Models over bandwidth-constrained datacentres. This project aims to investigate how to extract, adapt, and transfer those architectural paradigms into a downstream federated fine-tuning regime for Computer Vision tasks on highly heterogeneous edge devices.

To achieve this, the goal is articulated into three specific research objectives and structural contributions:

4.1.1 Partial-Replica Paradigm in FL

The first objective evaluates the viability of a partial-replica V-MoE within a real-world FL framework.

Device-side memory bottlenecks While partial replication was originally designed to reduce WAN bandwidth bottlenecks in datacentres, cross-device FL introduces a more immediate constraint: the device memory wall. Edge nodes exhibit various, often critical VRAM limitations and hardware heterogeneity; forcing a full replica of the entire expert pool onto a resource-constrained client is often physically impossible, acting as a binary threshold that would prevent highly constrained clients from participating.

To address these constraints, our methodology distributes only a specific subset of expert LoRA adapters to each client. We then leverage the *skip-token* mechanism to bypass missing remote computations, completely avoiding cross-node communication. This approach allows us to quantify the exact reductions in upload payload and local memory footprint compared to a full-replica baseline. Most importantly, we focus on assessing the core predictive trade-off: determining whether a partially

replicated model can maintain competitive accuracy when operating under extreme parameter sparsity and isolated local updates.

4.1.2 Static Expert Assignment

Fine-tuning a pre-trained MoE backbone implies that the network’s routers have already learned strong token-to-expert affinities. In a federated network characterised by highly non-IID (Independent and Identically Distributed) client data, these frozen routing preferences inevitably clash with local data skewness, leading to severely imbalanced expert activation. We investigate the structural challenges imposed by a *static* (fixed) expert placement strategy under these conditions.

Sparsity-aware aggregation Furthermore, we demonstrate how this severe heterogeneity necessitates specialised aggregation protocols. Specifically, we integrate the Sparsity-aware Model Aggregation (SaMA) scheme to ensure that global aggregations are strictly weighted by actual token routing counts, thus preventing the severe dilution of local LoRA matrices when experts are physically absent or unactivated on specific clients.

Exclusion of random assignment It is important to underline that the *random placement* strategy proposed in the original FoMoE framework (while highly effective as a regulariser for from-scratch pre-training) was deliberately excluded from the scope of this work. In a fine-tuning regime, experts already possess highly specialised, pre-trained feature representations; randomly reshuffling them across clients could destroy the alignment between the pre-trained weights and the downstream domain adaptation.

4.1.3 Dynamic Expert Reassignment and Trade-off Exploration

To address the rigidities and potential utilisation bottlenecks of static placement under non-IID data, the final objective of this project is the design, implementation, and critical evaluation of a distribution-aware *dynamic expert assignment* protocol.

Routing imbalances under non-IID In a cross-device federated environment, client data distributions are inherently heterogeneous ($P_i \neq P_j$). Because the V-MoE’s routing networks map tokens based on visual features, this non-IID conditioning naturally triggers highly skewed expert utilisation and *routing imbalances*. Local routers do not activate the expert pool uniformly; instead, specific clients generate severe load hot-spots on a small subset of experts that align with their local data domain, while leaving other expert modules entirely idle. Under a static assignment strategy, this behaviour creates a structural mismatch between where the data resides and where the required expert weights are physically hosted.

The skip-token penalty In addition, when accounting for our partial-replica setup, this mismatch introduces a severe operational penalty. When a client routes a token to an expert that is absent from its local hardware allocation (ghost expert), the system triggers the *skip-token* mechanism inherited from FoMoE. While bypassing the computation for these ghost experts successfully eliminates cross-node activation transfers over constrained networks, it inflicts a massive hidden cost in terms of model capacity. If a client’s local data distribution heavily relies on a missing expert, a substantial fraction of its total tokens will completely bypass the feed-forward layer. This structural dropping of tokens effectively deflates the active parameter capacity of the model during local training, severely limiting the network’s ability to extract specialised representations from that client’s data.

Dynamic assignment To mitigate these losses, this thesis finally introduces a *dynamic reassignment mechanism* as a more flexible, exploratory approach. Instead of keeping expert allocations fixed, the central server periodically redistributes the experts to match the different needs of individual clients based on their local data distributions. By doing so, we aim to directly align the learned specialisation of each expert with the distinct semantic properties of the clients’ datasets. The main goal here is not simply to force a performance boost, but rather to carefully investigate the actual trade-offs of moving parameters dynamically during fine-tuning. Specifically, we want to see if the practical advantages of reducing skip-token penalties and balancing the workload are ultimately cancelled out, or bottlenecked, by the subsequent bandwidth overhead and the difficulty of shifting pre-trained routing preferences in highly skewed, non-IID environments.

4.2 Architectural Analysis for Edge Deployment

Deploying deep learning models within a cross-device Federated Learning framework necessitates a paradigm shift in architectural design. Unlike centralised data-centre environments, where scaling laws are primarily constrained by training time and GPU clusters, edge deployment introduces strict, non-negotiable hardware constraints. The selection of a foundation model for such contexts cannot be driven solely by predictive performance; rather, it requires a judicious alignment with the physical limitations of the participating nodes.

The dual-sided MoE profile In this landscape, Mixture-of-Experts (MoE) architectures present an appealing, yet dual-sided, profile. The primary, well-established advantage of MoEs lies in their ability to decouple a model’s total parameter capacity from its active computational cost (FLOPs). By selectively routing inputs to only a specialised subset of experts, MoEs can vastly expand their knowledge base without inflating execution time or active compute operations. Furthermore, this conditional routing allows different experts to specialise in distinct features, effectively mitigating parameter interference during training.

The computational vs. memory dichotomy However, this computational sparsity introduces a *static memory* trade-off. While only a fraction of the network is active at any given step, the dynamic nature of the routing mechanism requires the entire parameter pool to be physically loaded in the device’s memory (VRAM). Consequently, the memory footprint scales directly with the size of the expert pool: the greater the number of experts introduced to enhance the model’s capacity, the more individual modules must be simultaneously available. In an edge environment, where VRAM is often a scarce resource, this dynamic requires careful consideration. The MoE model computes efficiently like a lightweight network, but its static memory requirements grow proportionally with the number of experts, mirroring the footprint of a larger dense architecture.

Hardware-constrained model dimensioning Consequently, the architectural blueprint must be highly contextualised, transforming the model selection process into a multidimensional optimisation problem. The base capacity of the network, the granularity of its tokenisation, and the total volume of the expert pool must be carefully calibrated to ensure that the memory peaks remain strictly below the operational limits of heterogeneous edge clients.

Systemic bias and representation In a federated ecosystem, an architecture that exceeds these local hardware thresholds does not merely suffer a drop in computational efficiency; it fundamentally precludes client participation. If the memory requirements are too stringent, the system inherently excludes lower-end devices from the federated loop. This exclusion introduces a severe statistical bias: the global model will only aggregate knowledge from high-tier hardware, failing to capture the true, diverse data distribution of the entire network. Ensuring hardware compliance is therefore not just an engineering necessity, but a fundamental prerequisite for robust model generalisation.

Navigating the structural trade-offs To prevent this representational collapse, the network’s dimensioning must often be guided by the “lowest common denominator” of the target edge environment. This requires a deliberate and granular tuning of the model’s structural hyperparameters. For instance, considering our V-MoE case study, increasing the patch size (and thereby reducing the token sequence length) quadratically cuts down the intermediate activations generated by the $O(N^2d)$ complexity of the self-attention mechanism, sacrificing microscopic spatial granularity to maintain an optimal training throughput. Similarly, the size of the expert pool and the dimensions of the shared backbone must be rigorously capped to prevent the static VRAM footprint from suffocating the device before the forward pass even begins.

Toward an empirical model choice Selecting a backbone for Federated Edge Learning therefore requires moving beyond standard datacentre scaling laws, and instead relies on a rigorous empirical profiling of memory peaks, execution latency, and communication payloads. It is precisely this pragmatic evaluation methodology

that motivates the adoption of the specific V-MoE configuration detailed in Section 4.4.

4.3 Fine-Tuning in Edge Federated Learning

While selecting an appropriately dimensioned backbone is the first step toward hardware compliance, the actual training phase introduces an entirely new set of systemic challenges. In standard datacentre fine-tuning, updating a model’s parameters requires storing not only the base weights and intermediate activations but also the computed gradients and the optimiser states. For widely used optimisers, like Adam [21], this effectively triples the static memory footprint of the trainable parameters, a demand that resource-constrained edge devices cannot afford.

The communication wall Beyond local memory limitations, Federated Learning is fundamentally bound by communication bandwidth. In a traditional Full Fine-Tuning (FFT) paradigm, each client must upload the entire updated model back to the central server at every communication round. For a multi-million parameter architecture like a V-MoE, transmitting the full gradient payload over standard edge networks (such as cellular or unstable Wi-Fi connections) introduces unacceptable latency, massive energy consumption, and high drop-out rates.

4.3.1 The PEFT Paradigm

To decouple the model’s representation power from the prohibitive costs of federated backpropagation, adopting a Parameter-Efficient Fine-Tuning (PEFT) strategy becomes a structural necessity rather than a mere optimisation. The core philosophy of PEFT in edge environments relies on freezing the vast majority of the pre-trained backbone, thereby nullifying the need to compute and store gradients and optimiser states for those layers.

LoRA as an architectural bridge Within this setting, as discussed in Section 3.2, Low-Rank Adaptation (LoRA) emerges as a highly synergistic approach for MoE architectures. Instead of updating the massive dense weight matrices, LoRA injects small, trainable low-rank decomposition matrices into the architecture alongside the frozen weights. By restricting the trainable parameter budget to these minimal adapters, the system achieves a dual benefit: it drastically collapses the peak VRAM required for the backward pass and reduces the federated communication payload by orders of magnitude. The architecture is thus adapted to the specific local domain by learning tiny, task-specific residual updates, transforming an otherwise impossible federated synchronisation into a lightweight, bandwidth-friendly operation.

4.4 Model Selection: V-MoE Small/32

Building upon the systemic prerequisites in Section 4.2, we now introduce the specific architecture deployed throughout the experimental pipeline: the Vision

4. Methodology: Partial-Replica V-MoE in Federated Learning

Mixture-of-Experts in the *Small/32* configuration (V-MoE-S/32). Following the nomenclature of standard Vision Transformers, the “Small” designation refers to the baseline parameter capacity of the dense architecture, while the “/32” suffix dictates a patch size resolution of 32×32 pixels.

4.4.1 Choice Rationale

The selection of the *Small/32* configuration for the V-MoE backbone is a deliberate decision dictated by the stringent hardware and communication constraints inherent to the target cross-device FL setting.

The parameter wall To empirically justify this selection, Table 4.1 provides a parameter comparison between the employed V-MoE-S/32 and the next available architectural scale, the V-MoE-B/16 (Base/16).

Metric	V-MoE-S/32	V-MoE-B/16	Ratio (B/16 $\div$ S/32)
Total parameters	57.00 M	300.97 M	5.28 $\times$
Dense (shared backbone)	23.40 M	74.30 M	3.17 $\times$
Sparse (expert pool)	33.60 M	226.68 M	6.75 $\times$
Params per single expert MLP	2.10 M	4.72 M	2.25 $\times$
Experts per MoE layer (E)	8	8	–
MoE layers	2	6	–
Sparse share of total	58.9%	75.3%	–

Table 4.1: **Parameter budget comparison between V-MoE-S/32 and V-MoE-B/16.** The B/16 variant is 5.28 $\times$ larger overall (300.97 M vs 57.00 M parameters), and its dense backbone alone (74.30 M) already exceeds the entire S/32 model; crucially, the mandatory frozen-backbone footprint that every client must load is 3.17 $\times$ smaller for the S/32 (23.40 M vs 74.30 M), motivating its selection for edge deployment.

A standard V-MoE model in a *Base/16* configuration, while highly capable, introduces an unsustainable architectural overhead for heterogeneous edge devices:

- The **overall parameter count** highlights a severe discrepancy between the two architectural tiers. The V-MoE-B/16 scales to a massive 301 M parameters, making it 5.28 $\times$ larger than the V-MoE-S/32 (57.00 M). Remarkably, the dense backbone of the B/16 variant alone (74.30 M) structurally exceeds the *entire* parameter budget of the complete S/32 model.
- The **dense backbone** represents the non-negotiable cost of the federation. While the volume of the transmitted updates is drastically reduced thanks to LoRA matrices, the frozen dense parameters must still be physically loaded into the device’s memory to perform the forward pass. The S/32 configuration fixes this mandatory VRAM floor at 23.40 M parameters, compared to the 74.30 M required by the B/16 variant (a 3.17 $\times$ reduction).

The VRAM and computational bottlenecks Beyond the static parameter count, the true binding constraint for edge-device deployment is the runtime memory

footprint required during a training step. To profile the two architectures under identical, conservative conditions, we measure them in a full fine-tuning regime (all parameters trainable, FP32, batch 32), which upper-bounds the cost of the lighter LoRA recipe deployed later; the results are reported in Table 4.2.

Hardware Metric	V-MoE-S/32	V-MoE-B/16	Ratio (B/16 $\div$ S/32)
<i>Memory Footprint</i>			
Static VRAM (weights)	0.21 GB	1.06 GB	5.00 $\times$
Peak VRAM (forward + backward)	0.74 GB	5.99 GB	8.14 $\times$
Optimiser state (AdamW)	0.45 GB	2.15 GB	4.79 $\times$
Peak VRAM (full training step)	1.09 GB	7.01 GB	6.42 $\times$
<i>Execution Performance</i>			
Batch Training Latency	10.48 $\pm$ 0.94 ms	86.97 $\pm$ 0.57 ms	8.30 $\times$
Per-image Latency	0.327 ms	2.718 ms	8.30 $\times$
Training Throughput	3,054 img/s	368 img/s	—

Table 4.2: **Empirical hardware profiling under a full fine-tuning regime (Batch = 32, FP32).** All parameters are trainable; the *optimiser state* accounts for the two AdamW moments over the full parameter set, and the *full training step* peak is the measured maximum over a complete forward, backward and optimiser update. These figures isolate the raw backbone cost and do not reflect the lighter LoRA/bfloat16 regime adopted for the federated fine-tuning of Section 4.5. The profiling confirms that only the S/32 is edge-viable: the full training-step peak diverges by 6.42 $\times$ (1.09 GB vs 7.01 GB) and the per-step latency by 8.30 $\times$ (10.48 ms vs 86.97 ms), collapsing throughput from 3054 down to just 368 images per second on the B/16.

As detailed in Table 4.2, empirical profiling reveals a severe hardware bottleneck that renders the B/16 architecture unviable for edge deployment. While the static weight allocations scale roughly linearly (0.21 GB for the S/32 versus 1.06 GB for the B/16), the peak VRAM of a complete training step — jointly accounting for weights, activations, gradients and the AdamW optimiser state — exposes a drastic gap. The S/32 model peaks at a highly manageable 1.09 GB, whereas the B/16 reaches 7.01 GB (a 6.42 $\times$ gap), driven by the massive intermediate activations generated by the finer /16 tokenisation. Notably, the gap is even sharper if the optimiser state is set aside: at the forward+backward stage alone, the activation-dominated footprint diverges by 8.14 $\times$ (0.74 GB versus 5.99 GB).

In addition to the memory wall, the B/16 architecture introduces a prohibitive operational penalty. Execution latency jumps from 10.48 ms to 86.97 ms per step (an 8.30 $\times$ increase), effectively crashing the training throughput from 3,054 down to an unviable 368 images per second.

Patch size implications While empirical profiling dictates the choice of the S/32 architecture, this hardware efficiency stems from a fundamental structural compromise. At a standard 224×224 input resolution, the B/16 model divides the image into 16×16 patches, yielding a dense sequence of 196 tokens. This fine-grained tokenisation captures minute local textures but triggers the $O(N^2d)$ [15] complexity of the self-attention mechanism, causing the prohibitive activation spike (a 5.99 GB forward+backward footprint, rising to 7.01 GB once the optimiser state is resident).

4. Methodology: Partial-Replica V-MoE in Federated Learning

Conversely, the S/32 configuration compresses the image into a minimal 7×7 grid (49 tokens). Because each token encompasses a receptive field four times larger, fine spatial details are inevitably smoothed out into macroscopic features.

In a standard datacentre setting, this loss of spatial granularity might be penalised. In a cross-device Federated Edge framework, however, hardware compliance is a strict binary threshold: if a model exceeds the local VRAM, the client simply cannot participate.

4.4.2 Architectural Overview

The V-MoE-S/32 model is composed as follows:

Transformer backbone The backbone is built upon a sparse Vision Transformer baseline consisting of $L = 8$ sequential Transformer blocks. The input image is divided into 32×32 pixel patches, which are linearly projected into an embedding space of dimension $d = 512$. Each standard dense block (Figure 4.1) features a Multi-Head Self-Attention (MHSA) module with 8 attention heads, followed by a dense Multi-Layer Perceptron (MLP) with a hidden dimension of 2048 ($4 \times d$).

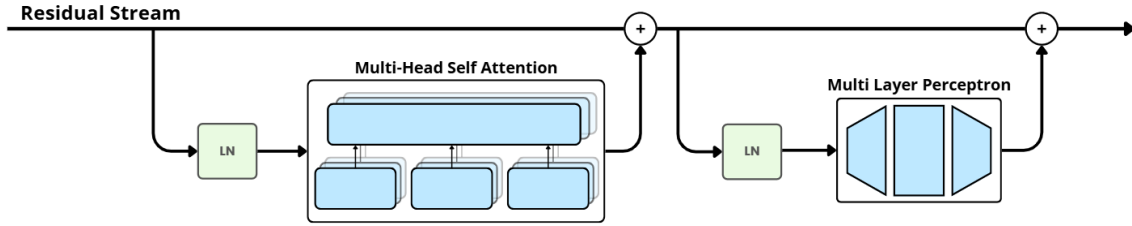


Figure 4.1: Dense transformer block

MoE configuration Following a *last-2* structural arrangement, the network’s density is interrupted in the final two odd blocks (specifically at indices 5 and 7), referred to throughout this work as *MoE layer 0* and *MoE layer 1*, respectively. In these specific layers, the standard dense MLP is replaced by a Mixture-of-Experts (MoE) layer (Figure 4.2) comprising:

- **Router:** A linear gating network that evaluates each token and assigns it to its top-2 most suitable experts ($k = 2$).
- **Expert Pool:** $E = 8$ independent experts, each mirroring the original MLP architecture ($512 \rightarrow 2048 \rightarrow 512$).

Pre-trained checkpoint The base weights were pre-trained from scratch on the **ImageNet-1k** (ILSVRC2012) [22] dataset. The upstream pre-training regimen spanned 300 epochs and incorporated *RandAugment* regularisation techniques to enhance visual robustness and generalisation. This foundational checkpoint achieves a Top-1 accuracy of 71.03% and a Top-5 accuracy of 89.36%, as empirically tested.

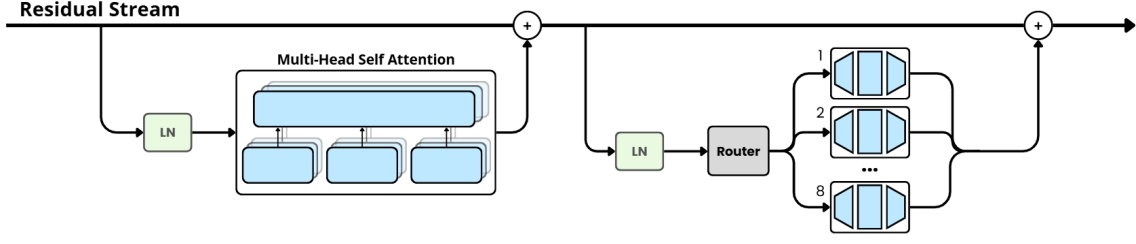


Figure 4.2: Sparse transformer block

4.5 Fine-Tuning Setting

To adapt the pre-trained V-MoE-S/32 model for downstream Federated Learning tasks under strict communication constraints, we employ a Parameter Efficient Fine-Tuning (PEFT) strategy based on LoRA matrices:

- **LoRA Injection:** Low-rank adaptation matrices, configured with rank $r = 8$ and a scaling factor $\alpha = 16$ (yielding a scaling ratio of 2.0 without dropout), are injected into the dense transformations of the network. Specifically, LoRA is applied to the Query, Key, Value, and Projection matrices across all 8 attention blocks, as well as to the inner linear layers (fc1 and fc2) of both the standard dense MLPs and all 16 experts within the MoE layers.
- **Full Fine-Tuning Components:** To calibrate the network to the specific downstream domain, the positional embeddings, the final classification head, and, crucially, the MoE routing networks are fully fine-tuned. Together, these components account for only a small fraction of the total network parameters. In particular, the MoE routers remain globally shared: they are uploaded, aggregated via standard FedAvg, and broadcast back to the clients, ensuring that the replicated experts coordinate coherently across all local distributions.
- **Frozen Backbone:** As a result, the vast majority of the network remains entirely frozen to prevent catastrophic forgetting and save computational resources. This includes the base multi-head attention weights, the base dense and expert MLP weights, the patch embedding projections, and all Layer Normalisation parameters.

Parameter and Communication Efficiency This sparse PEFT configuration radically alters the network’s trainable footprint and the resulting communication bottlenecks, as summarised in Table 4.3.

While the base architecture natively comprises 56.57 M parameters, the injection of the LoRA matrices expands the total capacity to 57.67 M. However, as detailed in Table 4.3, only 1.48 M parameters are actively updated during the local backward pass, representing a 2.56% of the total network capacity.

From a federated communication standpoint, the transmit payload is strictly defined by this trainable subset. Every communication round, each client exchanges exactly 1.48 M parameters with the central server. Using standard FP32 serialisation, this translates to a highly efficient upload payload of approximately 5.63 MB.

4. Methodology: Partial-Replica V-MoE in Federated Learning

Architectural Component	Parameter Count	Fraction of Total
Total Base Capacity	56.57 M	–
Total Capacity with LoRA	57.67 M	–
<i>Trainable Budget</i>		
LoRA Injection (Attention + MLPs)	1.10 M	1.91%
Full Fine-Tuning (Head, Routers, Embeddings)	0.38 M	0.65%
Total Trainable Parameters	1.48 M	2.56%
<i>Frozen Budget</i>		
Total Frozen Parameters	56.19 M	97.44%
Federated Communication Payload	Parameter Count	Data Size (FP32)
Upload per Client / Round	1.48 M	≈ 5.63 MB
Round-trip per Client / Round	2.95 M	≈ 11.26 MB

Table 4.3: **Parameter breakdown and communication payload analysis** for the V-MoE-S/32 architecture under the full-replica fine-tuning regime. LoRA restricts the trainable budget to just 1.48 M parameters—2.56% of the network—so that each client uploads only ≈ 5.63 MB per round rather than the full model. This establishes the full-replica communication cost against which the partial-replica strategies of the following sections are evaluated.

It is important to underline that in this full-replica baseline, the LoRA adapters for *all* 16 experts are systematically exchanged during every round, regardless of whether a specific expert was activated by the local data routing logic. This dynamic establishes the foundational communication cost against which subsequent partial-replica strategies will be evaluated.

4.6 Sparsity-aware Model Aggregation

As discussed in Chapter 3, standard FedAvg fails to account for the inherent sparsity of MoE architectures, leading to severe parameter dilution and the subsequent degradation of the aggregated global model. Consequently, integrating the Sparsity-aware Model Aggregation (SaMA) paradigm is strictly necessary for our heterogeneous federated setting. Translating this theoretical concept into practice, however, requires precise tracking mechanisms at the client level and a dual-track aggregation logic at the server level.

In standard FedAvg, the server requires only the model weights and the total dataset size n_k from each client k . Under the SaMA paradigm, the communication payload sent by the clients must also include a metadata dictionary containing the exact token routing statistics for their local experts. The implementation is therefore divided into two distinct phases.

4.6.1 Client-Side: Token Tracking Mechanism

During the local training phase, the client must transparently monitor the gating network to count how many tokens are assigned to each expert. In our PyTorch-

based implementation, this is achieved by injecting a tracking mechanism directly into the MoE router’s forward pass.

Algorithm 1 formalises this client routine. Upon selection, the client overwrites its local weights with the freshly received global parameters W_{tx} (L.2). It then performs the standard local optimisation across the configured local epochs (L.3–5); during every forward pass, the router accumulates the per-expert token counts into the routing-mass buffer m alongside the weight updates ΔW . Once training concludes, the client consolidates the total mass dispatched to each expert (L.6), implicitly flagging any never-selected expert with a zero entry. Finally, it transmits the tuple comprising the update ΔW , the local dataset size n , and the routing-mass vector m back to the server (L.7).

Algorithm 1 SaMA: client-side token tracking

```

1: procedure CLIENTUPDATE( $W_{\text{tx}}$ , Server)
2:    $W \leftarrow W_{\text{tx}}$ 
3:   for each local epoch do
4:      $\Delta W, m \leftarrow \text{train}(W)$  ▷ Accumulate per-expert routing mass
5:   end for
6:    $m \leftarrow \text{routing\_mass\_per\_expert}$  ▷  $\sum$  tokens dispatched to each expert
7:   send( $\Delta W, n, m$ , Server)
8: end procedure

```

4.6.2 Server-Side: Dual-Track Aggregation

Upon receiving the local updates and their associated token-tracking metadata, the central server initiates the synchronisation step. Because the V-MoE architecture comprises both dense components (e.g., attention layers) and sparse components (the MoE experts), the server must execute a *dual-track aggregation*.

The logic, detailed in Algorithm 2, evaluates each parameter of the model independently. After initialising the global model (L.2), each communication round begins by sampling the pool of available clients (L.4), broadcasting the current global model (L.5), and collecting the returned updates together with their dataset sizes and routing masses (L.6). The synchronisation then iterates over every parameter key k of the model (L.7). For each participating client, the algorithm branches on the nature of the parameter (L.8–13): if k belongs to an expert, the aggregation weight w_i is set to the routing mass that the client accumulated for that specific expert, floored at $\varepsilon \ll 1$ to neutralise idle contributions (L.9–10); otherwise, for dense backbone parameters, it falls back to the standard **FedAvg** weight given by the local dataset size n_i (L.11–12). The global increment is then obtained as the resulting weighted average across all clients (L.15) and applied to the model (L.16), before the converged weights are ultimately returned (L.19).

4. Methodology: Partial-Replica V-MoE in Federated Learning

Algorithm 2 SaMA: server-side dual-track aggregation

```

1: procedure SERVERAGGREGATION
2:    $W = \text{global\_model\_initializer}()$ 
3:   for each communication round do
4:      $\mathcal{S} = \text{client\_selector}()$ 
5:      $\text{send}(W, \mathcal{S})$ 
6:      $\{(\Delta W_i, n_i, \mathbf{m}_i)\}_{i \in \mathcal{S}} = \text{receive}(\mathcal{S})$ 
7:     for each parameter key  $k$  in the global model do
8:       for each client  $i \in \mathcal{S}$  do
9:         if  $k$  is an expert key (expert  $e$ , layer  $\ell$ ) then
10:           $w_i \leftarrow \max(\mathbf{m}_i[\ell, e], \varepsilon)$  ▷ Sparse track: SaMA
11:        else
12:           $w_i \leftarrow n_i$  ▷ Dense track: standard FedAvg
13:        end if
14:      end for
15:       $\Delta W_{\text{global}}[k] \leftarrow \frac{\sum_{i \in \mathcal{S}} w_i \cdot \Delta W_i[k]}{\sum_{i \in \mathcal{S}} w_i}$  ▷ Weighted average of the updates
16:       $W[k] \leftarrow W[k] + \Delta W_{\text{global}}[k]$ 
17:    end for
18:  end for
19:  return  $W$ 
20: end procedure

```

4.7 Partial-Replica Setup

In this section, we describe the design of the *partial-replica* setup, a configuration where each client is allocated only a specific subset of the global experts, denoted as E_{loc} . This approach is necessary because transmitting and storing the complete set of MoE experts requires substantial communication bandwidth and local memory, which directly contradicts the resource-constrained nature of edge devices.

4.7.1 Static Assignment

The most fundamental strategy to distribute the partial expert pools is the *static assignment* (or fixed allocation), directly inherited by the FoMoE framework. In this scenario, the specific subset of E_{loc} experts assigned to a client i is determined prior to the training phase, and remains entirely immutable across all communication rounds. Consequently, the server must maintain a registry of these assignments. During the broadcast phase, instead of transmitting the full global model, the server dynamically filters the parameter dictionary: it transmits the shared dense backbone to all selected clients, but pushes only the designated subset of experts to each specific device.

Algorithm 3 integrates this filtering into the federated loop. The total and local expert budgets E_{tot} and E_{loc} are provided as inputs, together with the number of clients N . Before training starts, the server initialises the global model W and draws the immutable per-client expert subsets $\{\mathcal{E}_1, \dots, \mathcal{E}_N\}$ (L.2–3). At each round, once

the clients are sampled (L.5), the broadcast phase is specialised per client: rather than shipping the entire model, the server assembles a tailored payload combining the shared dense backbone W_{dense} with only the experts $W_{\mathcal{E}_i}$ assigned to that client, and transmits it (L.7–8). After collecting the local updates (L.10), the aggregation loops over every parameter key (L.11) and first restricts the client set to $\mathcal{S}_k$, i.e. only those clients that actually hold parameter k (L.12). The SaMA dual-track rule of Algorithm 2 is then reused verbatim over this restricted subset (L.13), so that the weighted-average denominator is correctly adjusted for sparsely replicated experts before the update is applied to the global model (L.14).

Algorithm 3 Static partial-replica

Require: E_{tot} : total number of experts per MoE layer, E_{loc} : number of partial experts replicated per MoE layer, N : total number of clients

```

1: procedure STATICROUNDEXECUTION
2:    $W \leftarrow \text{global\_model\_initializer}()$ 
3:    $\{\mathcal{E}_1, \dots, \mathcal{E}_N\} \leftarrow \text{AllocateExperts}(E_{\text{tot}}, E_{\text{loc}}, N)$ 
4:   for each communication round do
5:      $\mathcal{S} \leftarrow \text{client\_selector}()$ 
6:     for each client  $i \in \mathcal{S}$  do
7:        $W_{\text{target}} \leftarrow W_{\text{dense}} \cup W_{\mathcal{E}_i}$ 
8:        $\text{send}(W_{\text{target}}, i)$ 
9:     end for
10:     $\{(\Delta W_i, n_i, \mathbf{m}_i)\}_{i \in \mathcal{S}} \leftarrow \text{receive}(\mathcal{S})$ 
11:    for each parameter key  $k$  in the global model  $W$  do
12:       $\mathcal{S}_k \leftarrow \{i \in \mathcal{S} \mid k \in \text{keys}(\Delta W_i)\}$ 
13:       $\Delta W_{\text{global}}[k] \leftarrow \text{SaMA\_DualTrack}(k, \mathcal{S}_k)$ 
14:       $W[k] \leftarrow W[k] + \Delta W_{\text{global}}[k]$ 
15:    end for
16:  end for
17:  return  $W$ 
18: end procedure
    
```

Remark on communication overhead and LoRA For notational simplicity, the pseudocode in the previous algorithms uses generic variables (e.g., W , W_{target} , and ΔW_i) to represent the entire model. In practice, however, the actual communication is highly optimised. As already detailed before, most of the pre-trained backbone parameters are strictly frozen. These heavy static weights are transmitted by the server only once at initialisation (or loaded directly from a local checkpoint) and are never exchanged again. Throughout the training rounds, the communication payloads contain only the lightweight trainable parameters: the LoRA matrices, alongside the full fine-tuned parameters (e.g., MoE routers).

4.7.2 Practical Deployment

To translate the theoretical static assignment into a functional federated deployment, the architecture relies on three interconnected software-engineering mecha-

4. Methodology: Partial-Replica V-MoE in Federated Learning

nisms. These operate seamlessly across the communication rounds without requiring structural alterations to the underlying PyTorch network graphs. The core design principle enabling this flexibility is a flat, string-based parameter namespace: every trainable tensor is identified by a globally unique dotted key that encodes both its transformer block and its expert index (e.g., `blocks.5.mlp.experts.3.fc1.lora_A`). This allows the heterogeneous federation to be expressed, filtered, and merged entirely through dictionary operations.

Server-side filtering and broadcast The static assignment is materialised before the first training round and is held in a frozen registry. This data structure maps each client to its assigned subset of E_{loc} experts for every MoE layer. Crucially, the random seed used to draw these subsets is varied across different transformer blocks to maximise the federation’s collective coverage of the global expert space.

During the broadcast phase, the server leverages this registry to construct a highly customised payload for each selected client. Rather than transmitting the full E_{tot} expert pool, the server iterates over the global PyTorch `state_dict`. It extracts the shared dense backbone (such as attention layers, normalisation layers, and MoE routers) and explicitly filters the sparse components, retaining only the keys corresponding to the specific experts assigned to the target client. This server-side filtering strictly enforces the communication constraints of the partial-replica setup, ensuring that the downstream bandwidth consumption scales with E_{loc} rather than E_{tot} .

Client-side constrained routing Upon receiving the tailored `state_dict`, the client loads the partial model. As previously discussed, the gating network is deliberately instantiated with the full routing dimension (E_{tot}) instead of being reduced to the local pool size (E_{loc}). This allows the router to evaluate the entire global expert space during the forward pass. Thus, the system can accurately track how often the router attempts to select the missing modules, effectively measuring the demand for the ghost experts.

Ghost expert and skip-token implementation During the forward pass, the routing mechanism operates in two distinct logical steps to preserve both metric integrity and computational validity. First, the gating network computes the raw routing logits for the entire set of E_{tot} experts. Before any physical constraint is applied, the unconstrained top predictions are evaluated against a boolean buffer, `local_expert_mask` (where entry i is `True` if expert i is locally available). If the router’s preferred choice falls outside the local pool, this event is recorded in the local telemetry. This tracking is essential for quantifying the ghost experts’ activation frequency, providing crucial metrics on how badly the static assignment matches the client’s non-IID data distribution.

Once the metrics are safely collected, the physical routing constraint is enforced to implement the *skip-token* mechanism for ghost experts. First, the gating network applies the *softmax* and *Top-k* operations over the unconstrained global distribution (logits). If a token’s selected slot lands on a ghost expert, the corresponding routing probability is explicitly zeroed out post-selection, without renormalisation. Consequently, the token’s residual stream simply loses that fractional contribution, effectively skipping the unavailable expert’s computation. Throughout the local epochs, the client accumulates the valid routing probabilities dispatched to the owned E_{loc} experts to generate the routing mass vector $\mathbf{m}$. At the end of the round, the client uploads its updated parameter dictionary alongside the tracking metadata. An overview of the code implementing this mechanism can be seen in Code 4.1

```

1 class GhostMoELayer(nn.Module):
2     ...
3     def forward(self, input_patch):
4         # Step 1: Raw logits over the entire global space ( $E_{\text{tot}}$ )
5         gate_logits = self.router.dense_linear(input_patch)
6
7         # Step 2: Ghost telemetry (Unconstrained choice)
8         # We track the primary preference to capture the strongest signal of
9         # ghost expert demand
10        unconstrained_top1 = gate_logits.argmax(dim=-1)
11        self.ghost_top1_count += (~self.local_expert_mask[unconstrained_top1]).sum()
12
13        # Step 3: Unconstrained Routing (Softmax & Top-K)
14        gate_probs = gate_logits.softmax(dim=-1)
15        exp_probs, exp_indices = gate_probs.topk(self.router.top_k, dim=-1)
16
17        # Step 4: True Skip-Token (Zero out ghost weights post-selection)
18        is_local_slot = self.local_expert_mask[exp_indices]
19        exp_probs = exp_probs * is_local_slot.to(exp_probs.dtype)
20
21        # Step 5: Dispatch & SaMA Mass Accumulation
22        if self.training:
23            self.routing_mass.scatter_add_(
24                0, exp_indices.flatten(), exp_probs.flatten().double()
25            )
26
27        return self._dispatch_to_local_experts(input_patch, exp_probs, exp_indices)

```

Code 4.1: **Ghost-expert routing with post-selection masking.** The gate initially scores and selects from the full pool (E_{tot}) to quantify the ghost demand. Subsequently, the routing weights for selected ghost experts are zeroed out without renormalisation, effectively skipping the token computation for those slots.

4.8 Dynamic Expert Assignment

A central objective of this work is to evaluate the partial replication strategy under both static and dynamic conditions. To this end, we introduce the *Dynamic Expert Re-assignment* mechanism, designed to mitigate the limitations of fixed expert allocations in highly non-IID settings. In this section, we first outline the general methodology and the constraints imposed on the swapping process to preserve model stability. Next, we define the metric governing the expert selection for reassignment. Finally, we formalise the entire process through pseudo-code, detailing the respective roles of the clients and the central server.

4.8.1 Methodology

To address the drawbacks of a fixed expert topology, we propose a server-orchestrated reassignment methodology built upon three core concepts:

The bottleneck of static routing Under highly non-IID data distributions, a static, random assignment of the E_{loc} experts can severely hinder model convergence. Although the skip-token mechanism successfully mitigates the latency and bandwidth costs of cross-site activations, it inherently discards computations when a client’s local data relies heavily on experts outside its static pool.

The evaluation window and swap cycle Rather than forcing clients to train with suboptimal experts, the system periodically evaluates their local demand for ghost experts. This reassignment is managed entirely by the central server, which aggregates *telemetry feedback* over consecutive *evaluation windows*. Each window spans t_{assign} communication rounds. During this period, clients accumulate and transmit their demand metrics. At the end of the window, a designated *swap round* takes place: the server analyses the aggregated feedback and orchestrates the necessary expert exchanges. Consequently, if a client consistently requires a specific ghost expert, that expert is eventually integrated into its local pool.

Semantic alignment objective Ultimately, this dynamic assignment aims to align each client’s local expert pool with the unique characteristics of its dataset. By exposing experts to more semantically coherent local inputs, we aim to enhance their specialisation and overall representational quality.

4.8.2 Swap Rules

To guarantee the overall stability of the training process and prevent phenomena such as the expert ping-pong effect, the dynamic assignment algorithm is governed by a set of server-evaluated rules.

Temporal constraints

- **Periodicity** ($t_{\text{assign}} = 5$): As previously introduced, reassignments occur exclusively at the end of each evaluation window, empirically set to 5 rounds. This prevents reactive thrashing caused by the statistical noise of a single communication round.
- **Warm-up phase**: No swaps are allowed during an initial warm-up phase, corresponding to the first 10% of the compute budget (5 rounds in our 50-round scenario). This delay grants the pre-trained routing network and the randomly initialised adapters sufficient time to stabilise their initial learning trajectories.
- **Final convergence**: To guarantee that the swapped experts have sufficient time to converge, all reassignments are strictly frozen after a predefined deadline (set at round 35 out of 50). This allocates the final portion of the training budget purely to local adaptation.

Capacity-Preserving Swaps For any given MoE layer, the server authorises at most one migration action per client per evaluation cycle. Because clients are initialised at their maximum local expert capacity (E_{loc}), this action strictly takes the form of a *capacity-preserving swap* (one eviction paired with one acquisition). This single-swap limit is a deliberate design choice serving three fundamental objectives:

- It ensures a **constant resource footprint**. By enforcing a zero-sum exchange, the total number of active experts on any client remains invariant, ensuring that VRAM consumption and communication payloads remain exactly constant for a clean, cost-isolated comparison against static baselines.
- It acts as a **cold-start mitigation** mechanism. Restricting the turnover to a single expert prevents massive injections of gradient noise during server-side aggregation, allowing the network to smoothly assimilate the new topology.

- It reflects an **incremental greedy adaptation** strategy, executing only the single most advantageous move per cycle to avoid destabilising shifts.

Structural Boundaries The selection of both the candidate (e_{cand}) and the victim (e_{victim}) relies on a specific demand evaluation score, $\hat{s}(e)$, which is formally defined in the following section. Once proposed based on these scores, the exchange must still follow these structural rules:

- **Admission and Replacement Margin:** The candidate must initially surpass an admission threshold (τ_{in}). Moreover, the candidate’s demand metric must strictly dominate the victim’s score by a *replacement margin* (τ_{swap}), effectively satisfying:

$$\hat{s}(e_{\text{cand}}) \geq \hat{s}(e_{\text{victim}}) + \tau_{\text{swap}} \quad (4.1)$$

- **Cooldown Period:** A newly acquired expert is protected from eviction for a fixed number of rounds (set at 10). This prevents rapid “add-then-evict” cycles, giving the gating network time to recognise the new expert and allowing its LoRA adapters to specialise on local data.
- **Global Diversity Floor** ($R_{\text{min}} = 2$): To prevent any expert from being entirely discarded by the federation, the server rejects evictions that would reduce an expert’s global replication count below R_{min} . A fail-safe mechanism unconditionally protects any expert with a global coverage of 2.
- **Global Replication Ceiling** ($R_{\text{max}} = 6$): To avoid a scenario where a globally prominent expert monopolises the network—thereby undermining the specialisation inherent to the MoE architecture—the server blocks the assignment of any candidate already distributed to R_{max} clients.

In scenarios where multiple clients target the same candidate or victim, the server arbitrates these conflicts by prioritising clients exhibiting the highest local scores, effectively enforcing these global capacity boundaries.

4.8.3 Swap Evaluation Metric

The decision of *which* expert to swap is driven by a continuous metric computation pipeline:

Client-Side: Local Demand and Temporal Smoothing During each federated round, the client’s gating network calculates the full, unconstrained routing distribution $G_e(x)$ over all E_{tot} experts (both owned and ghost). The client derives two distinct signals per expert aggregated over all batches in the round:

- **Cumulative Demand:** The mean routing probability across all N tokens processed in the round, capturing the overall volume of requests (generalist signal):

$$s_c^{\text{cumul}}(e) = \frac{1}{N} \sum_{i=1}^N G_e(x_i) \quad (4.2)$$

- **Specific Demand:** The peak single-token probability observed, preserving the signal for highly specialised experts that are rarely routed to but selected with extremely high confidence for specific hard tokens (specialist signal):

$$s_c^{\text{specific}}(e) = \max_{1 \leq i \leq N} G_e(x_i) \quad (4.3)$$

4. Methodology: Partial-Replica V-MoE in Federated Learning

These signals are linearly combined into a raw local importance score using a trade-off parameter λ (set to 0.8):

$$s_c(e) = \lambda s_c^{\text{cumul}}(e) + (1 - \lambda) s_c^{\text{specific}}(e) \quad (4.4)$$

Explanation of the dual-signal approach The decision to combine these two orthogonal signals is a design choice addressing the specific challenges of our non-IID, partial-replica setting. Relying on either metric in isolation introduces predictable failure modes:

- **Cumulative-only ignores the tail:** Averages naturally dilute rare signals. A specialist expert crucial for a small subset of data (e.g., a 15% off-theme tail) will have a low mean score and get discarded. Additionally, averages are highly susceptible to the router’s global popularity bias, pushing clients toward dominant experts rather than promoting local specialisation.
- **Specific-only chases noise:** Relying solely on peak activation makes the system vulnerable to outliers. It risks overvaluing a poor expert based on a single noisy token, while undervaluing reliable experts that provide steady utility without massive spikes.

By blending these two signals, the metric primarily relies on generalist usage to maintain high accuracy on the bulk of the data, while reserving a 20

Temporal Smoothing To filter out single-round router noise and prevent abrupt swaps based on temporary data spikes, the raw score $s(e)$ is not used directly. Instead, it is smoothed over the federated timeline using an *Exponential Moving Average* (EMA). This technique acts as a “temporal stabiliser”, blending the expert’s historical importance with its newly computed demand:

$$\bar{s}_c(e) \leftarrow \beta \bar{s}_c(e) + (1 - \beta) s_c(e) \quad (4.5)$$

By setting the decay factor to $\beta = 0.7$, the system essentially retains 70% of the expert’s historical score while updating it with 30% of the current round’s information. This inertia guarantees that an expert must demonstrate a sustained trend of high or low demand to trigger a swap, rather than reacting to a single anomalous batch.

Notably, when the server changes an expert’s role (from owned to ghost or vice versa), the client explicitly resets its EMA history to prevent stale metrics from contaminating the post-swap evaluation.

Server-Side: Prior Correction and Target Selection Relying solely on the raw local routing mass $\bar{s}(e)$ would cause all clients to converge toward the globally preferred experts dictated by the frozen pre-trained router, leading to severe replication ceiling collisions. To neutralise this global bias, we introduce a *Prior Correction* mechanism.

The server first aggregates the federation-wide routing mass to compute a global router prior, $\text{prior}(e)$. It then applies a correction factor to the client’s smoothed score:

$$\hat{s}_c(e) = \bar{s}_c(e) - \alpha (\text{prior}(e) - \overline{\text{prior}}) \quad (4.6)$$

where $\alpha = 0.5$ dictates the correction strength, and $\overline{\text{prior}}$ is the dynamically computed arithmetic mean of the global prior across all E_{tot} experts in the layer. This corrected metric, $\hat{s}_c(e)$, isolates the truly client-specific demand by penalising globally over-represented experts and boosting locally demanded specialised experts.

Finally, the server uses this single, unified vector $\hat{s}_c(e)$ to identify both the candidate and the victim for the swap operation:

- **Candidate-In:** Selected by *maximising* $\hat{s}_c$ over the client’s non-owned (ghost) experts, provided it clears the admission threshold.
- **Victim-Out:** Selected by *minimising* $\hat{s}_c$ over the client’s owned experts, excluding those protected by the cooldown mechanisms.

4.8.4 Reassignment Algorithm

The complete end-to-end logic executed by the federated system is formalised in two distinct phases: the local demand tracking performed by each client and the global arbitration and reassignment executed by the central server.

The client-side demand tracking is reported in Algorithm 4. At the start of the round the client records the total number of processed tokens N_t (L.2) and, throughout the local training pass, accumulates the standard aggregation payload of Algorithm 1 (the weight update ΔW and the per-expert routing mass $\mathbf{m}$) while additionally capturing the full unconstrained routing distribution $G_e(x_i)$ over all E_{tot} experts, including the ghost ones (L.3–6). It then post-processes these statistics expert by expert (L.7–12): it computes the generalist cumulative-demand signal $s_c^{\text{cumul}}(e)$ (L.8) and the specialist peak-demand signal $s_c^{\text{specific}}(e)$ (L.9), blends them through the trade-off parameter λ into the raw importance score $s_c(e)$ (L.10), and folds this value into the temporal EMA score $\bar{s}_c(e)$ (L.11). Finally, the client ships the complete round payload to the server (L.13): besides the smoothed demand vector $\bar{s}_c$ that drives the reassignment logic, it also transmits the trained model update ΔW , the local dataset size n , and the routing mass $\mathbf{m}$ required by the SaMA dual-track aggregation.

Algorithm 4 Dynamic reassignment: client-side demand tracking

Require: λ : trade-off parameter, β : EMA decay factor

```

1: procedure CLIENTDEMANDTRACKING( $W$ , Server)
2:    $N_t \leftarrow \text{total\_tokens\_in\_round}$ 
3:   for each local epoch do
4:      $\Delta W, \mathbf{m} \leftarrow \text{train}(W)$  ▷ Standard pass
5:      $G_e(x_i) \leftarrow \text{unconstrained\_routing\_probabilities}()$ 
6:   end for
7:   for each expert  $e \in E_{\text{tot}}$  do
8:      $s_c^{\text{cumul}}(e) \leftarrow \frac{1}{N_t} \sum_{i=1}^{N_t} G_e(x_i)$ 
9:      $s_c^{\text{specific}}(e) \leftarrow \max_{1 \leq i \leq N_t} G_e(x_i)$ 
10:     $s_c(e) \leftarrow \lambda \cdot s_c^{\text{cumul}}(e) + (1 - \lambda) \cdot s_c^{\text{specific}}(e)$  ▷ Raw importance score
11:     $\bar{s}_c(e) \leftarrow \beta \cdot \bar{s}_c(e) + (1 - \beta) \cdot s_c(e)$  ▷ Update temporal EMA score
12:  end for
13:  send( $\Delta W, n, \mathbf{m}, \bar{s}_c$ , Server)
14: end procedure

```

Algorithm 5 orchestrates the server-side arbitration. The routine first enforces the temporal constraints, returning the topology untouched whenever the round is not a swap round, still lies in the warm-up phase, or has passed the freezing deadline (L.2–4). Otherwise, it processes each MoE layer independently (L.6). *Phase 1* aggregates the federation-wide routing mass into the global prior and its scalar mean (L.7–8), and subtracts the

4. Methodology: Partial-Replica V-MoE in Federated Learning

prior-correction term from every client score to obtain the debiased metric $\hat{s}_c(e)$ (L.9–14). *Phase 2* then generates at most one proposal per MoE layer per client (L.15–24): it selects the highest-scoring ghost expert as candidate under the replication ceiling (L.17) and the lowest-scoring owned expert as victim outside the cooldown set (L.18), skips the client whenever the candidate fails the admission threshold (L.19) or the eviction would breach the diversity floor (L.20), and registers the swap only when the replacement margin τ_{swap} is satisfied (L.21–23). Finally, *Phase 3* resolves the concurrent conflicts by enforcing the global capacity boundaries (L.26) and refreshes the cooldown registries (L.27), before the updated topology $\mathcal{A}_{\text{new}}$ is broadcast for the next round (L.29).

Algorithm 5 Dynamic reassignment: server-side orchestration

Require: Current expert topology $\mathcal{A}$, smoothed demand vectors $\{\bar{s}_c\}_{c \in \mathcal{S}}$, current round R .

Require: Hyper-parameters: t_{assign} (period), τ_{in} (admission), τ_{swap} (margin), R_{min} , R_{max} , α .

```

1: procedure SERVERSWAPARBITRATION( $R, \mathcal{A}, \{\bar{s}_c\}_{c \in \mathcal{S}}$ )
2:   if  $R \bmod t_{\text{assign}} \neq 0$  or IsWarmup( $R$ ) or IsPastDeadline( $R$ ) then
3:     return  $\mathcal{A}$ 
4:   end if
5:    $\mathcal{P} \leftarrow \emptyset$ 
6:   for each MoE layer  $\ell$  do
7:      $\text{prior} \leftarrow \frac{1}{|\mathcal{S}|} \sum_{c \in \mathcal{S}} \bar{s}_c$ 
8:      $\overline{\text{prior}} \leftarrow \frac{1}{E_{\text{tot}}} \sum_{e=1}^{E_{\text{tot}}} \text{prior}(e)$ 
9:     Phase 1: Metric Correction
10:    for each client  $c \in \mathcal{S}$  do
11:      for each expert  $e \in \{1, \dots, E_{\text{tot}}\}$  do
12:         $\hat{s}_c(e) \leftarrow \bar{s}_c(e) - \alpha \cdot (\text{prior}(e) - \overline{\text{prior}})$ 
13:      end for
14:    end for
15:    Phase 2: Local Proposals
16:    for each client  $c \in \mathcal{S}$  do
17:       $e_{\text{cand}} \leftarrow \arg \max_{e \notin \mathcal{E}_c^\ell, \text{cov}(e) < R_{\text{max}}} \hat{s}_c(e)$ 
18:       $e_{\text{victim}} \leftarrow \arg \min_{e \in \mathcal{E}_c^\ell \setminus \text{cooldown}} \hat{s}_c(e)$ 
19:      if  $\hat{s}_c(e_{\text{cand}}) < \tau_{\text{in}}$  then continue
20:      if  $\text{cov}(e_{\text{victim}}) \leq R_{\text{min}}$  then continue
21:      if  $\hat{s}_c(e_{\text{cand}}) \geq \hat{s}_c(e_{\text{victim}}) + \tau_{\text{swap}}$  then
22:         $\mathcal{P} \leftarrow \mathcal{P} \cup \{\text{SWAP}(c, \ell, e_{\text{victim}} \rightarrow e_{\text{cand}})\}$ 
23:      end if
24:    end for
25:    Phase 3: Global Arbitration
26:     $\mathcal{A}_{\text{new}} \leftarrow \text{ArbitrateConflicts}(\mathcal{P}, \mathcal{A}, R_{\text{max}})$ 
27:    UpdateCooldownRegistries( $\mathcal{A}_{\text{new}}$ )
28:  end for
29:  return  $\mathcal{A}_{\text{new}}$ 
30: end procedure

```

Global Arbitration and Concurrent Conflicts As detailed in Phase 3, the `ArbitrateConflicts` function explicitly evaluates both the replication ceiling ($R_{\max}$) and the diversity floor ($R_{\min}$) to safely resolve simultaneous reassignment requests. Because the local proposals generated in Phase 2 rely on a frozen coverage snapshot to prevent intra-round drift, multiple clients might independently target the same experts. This creates two distinct collision risks: concurrent acquisitions could push a popular expert above the $R_{\max}$ limit, while concurrent evictions could drop a vulnerable expert below the $R_{\min}$ survival threshold.

To prevent this, the server aggregates all valid proposals in $\mathcal{P}$ and enforces both systemic boundaries post-hoc. In the event of a collision, the server arbitrates the conflict by sorting the requests based on the local demand metric ($\hat{s}_c$). Limited acquisition slots are granted strictly to the clients exhibiting the highest demand for the candidate, while limited eviction allowances are granted to those with the absolute lowest scores for the victim. Any excess proposals that would violate the global capacity constraints are safely discarded.

Chapter 5

Experimental Results

Contents

5.1	Experimental Setup	60
5.1.1	Flower Framework	60
5.1.2	Federated Configuration	61
5.1.3	Datasets and Non-IID Partitioning	62
5.2	Evaluation Baselines	65
5.2.1	Centralised V-MoE Upper Bound	65
5.2.2	Full-Replica Federated V-MoE	67
5.2.3	Dense Federated ViT-S/32	67
5.2.4	Results	67
5.3	Partial-Replica: Static and Dynamic	69
5.3.1	Premises and Expectations	69
5.3.2	Replication Strategies at Initialisation	70
5.3.3	Expected Outcomes	71
5.4	Static Assignment	71
5.4.1	Model Performance	71
5.4.2	Expert Specialisation	72
5.4.3	Ghost Activations	76
5.4.4	System Metrics	78
5.4.5	The accuracy–system trade-off	80
5.5	Dynamic Reassignment	81
5.5.1	Model Performance	81
5.5.2	Expert Specialisation	82
5.5.3	Ghost Activations	84
5.5.4	Swaps Analysis	85
5.5.5	The Cost of Dynamism	86
5.6	Summary of the Findings	90

5.7	Conclusions and Future Work	93
------------	--	-----------

5. Experimental Results

This chapter presents the experimental validation of the methodology developed in Chapter 4, evaluating the partial-replica V-MoE along the accuracy, routing, memory, and communication axes. It first describes the experimental environment—the Flower-based federation, the fine-tuning configuration, and the two benchmarks, CIFAR-100 and Office-Home, with the partitioning schemes that induce the non-IID regimes under study—and then establishes the centralised, full-replica, and dense baselines against which every result is measured.

The core of the chapter follows the two regimes of the proposed system. The *static* analysis quantifies the accuracy cost and the system savings of a fixed partial-replica assignment, together with the routing behaviour that explains them. The *dynamic* analysis then evaluates the expert reassignment protocol, measuring its effect on accuracy, stability, and communication overhead. A visual summary of the findings and the conclusions, with the directions for future work, close the chapter.

5.1 Experimental Setup

This first section establishes the common experimental ground shared by every run of the campaign: the Flower framework that orchestrates the federation, the federated fine-tuning configuration, and the benchmark datasets together with the partitioning procedures that induce the two non-IID regimes under study.

5.1.1 Flower Framework

Flower (Beutel et al. [10], Figure 5.1) is an open-source framework designed for defining and running federated learning workloads, from large-scale cloud deployments to edge devices with limited resources.



Figure 5.1: Flower logo

Throughout the experimental phase, all Federated Learning workloads are entirely orchestrated by the Flower framework, capitalising on its robust deployment capabilities. In particular, the framework’s inherent modularity was utilised to implement custom, sparsity-aware aggregation strategies (as detailed in Section 4.6). This allowed the central server to seamlessly process dynamically shifting topologies and heterogeneous client updates, selectively aggregating only the subset of trainable parameters without enforcing symmetric model constraints across the edge network.

Framework-agnostic orchestration A distinguishing feature of Flower is its framework-agnostic design, which seamlessly integrates with major machine learning libraries such as PyTorch, TensorFlow, and JAX. Furthermore, it provides a highly extensible architecture for federated aggregation. While offering standard baselines like FedAvg, it

allows researchers to easily implement custom server-side strategies. This modularity is particularly crucial for complex synchronisation loops, where only a highly targeted subset of trainable parameters must be extracted, aggregated, and broadcast back to the edge nodes at each communication round.

Cross-platform deployment and interoperability Furthermore, the framework’s reach extends well beyond traditional server environments, natively supporting the deployment of federated workloads across cloud providers, mobile platforms, and embedded edge computing units. This high degree of platform independence ensures seamless interoperability across highly heterogeneous operating systems and hardware specifications. From a practical standpoint, this architectural robustness minimises the engineering effort required to transition experimental research into functional production systems. Despite these advanced capabilities, the underlying complexity remains encapsulated within an accessible interface, suitable for users across varying levels of technical expertise.

5.1.2 Federated Configuration

All experiments are conducted based on the following federation setting:

Client pool and participation The federated network is simulated over a fixed pool of $N = 8$ *clients*. To ensure that any variations in performance are strictly caused by the heterogeneous expert allocation, rather than the randomness of client sampling, we enforce a *full participation* regime. Thus, the strategy dictates that all 8 clients are selected and contribute to the global update at every communication round.

Model architecture The underlying model is a *Vision Mixture-of-Experts* (V-MoE), specifically the V-MoE-S/32 variant already introduced and detailed in Section 4.4.

Fine-tuning dynamics The fine-tuning process proceeds for $R = 50$ synchronous communication rounds. When selected, a client performs $E = 1$ *local epoch*, representing a full pass over its allocated data shard. Each client considers a *local batch size* of 64.

Loss function and regularisation The training objective is the *label-smoothed cross-entropy* ($\epsilon_{ls} = 0.1$), which mitigates overfitting by discouraging over-confident predictions. The MoE router operates *deterministically*, so no noise is added to the logits. Regularisation consists solely of decoupled weight decay (0.05, applied to two-dimensional weight matrices only); LoRA dropout is disabled.

Optimiser Optimisation is driven by the *AdamW* optimiser under *bfloat16* precision. To accommodate the diverse roles of the trainable parameters, we implement a discriminative learning-rate scheme: 3×10^{-4} for the LoRA adapters, 5×10^{-4} for the classification head, and a deliberately conservative 1×10^{-5} for the MoE routers, ensuring the stability of the globally shared gating function. Within each communication round the optimiser is re-initialised and follows a cosine schedule with a linear warmup over the first 10% of the client’s local steps, decaying to a floor of 10^{-6} by the end of the local epoch.

Evaluation protocol To ensure a comprehensive performance assessment, the model’s evaluation is conducted on both server and client sides. The primary global benchmark is derived via *centralised server-side evaluation*: at the end of each round, the

5. Experimental Results

freshly aggregated parameters are loaded into the server-resident registry model, which is evaluated against a held-out global validation set, yielding the reported Top-1 accuracy (Acc@1) and Top-5 accuracy (Acc@5) as the main performance metrics. Concurrently, a *federated client-side evaluation* is executed across all participating nodes on their respective local splits, providing a granular mechanism for per-node diagnostic tracking and local convergence monitoring.

5.1.3 Datasets and Non-IID Partitioning

This section first defines the two axes of non-IID heterogeneity targeted in this work, then introduces the benchmark datasets used to instantiate them, and finally details the deterministic procedures that distribute the dataset samples across the federated client pool. Crucially, every partitioning procedure is governed by fixed random seeds, ensuring that the induced heterogeneity is strictly reproducible and consistent across all experimental baselines.

Datasets involved The experimental evaluation leverages two heavily benchmarked vision datasets, chosen specifically for their distinct structural properties.

CIFAR-100 The *CIFAR-100* dataset [23] comprises 60,000 colour images at a native 32×32 resolution, evenly distributed across 100 *fine-grained classes* (500 training and 100 testing images per class). Its defining feature is a two-level semantic taxonomy: the 100 fine classes are grouped into 20 *coarse superclasses* (e.g., *aquatic mammals*, *vehicles 1*, *flowers*), each containing five semantically related categories. For the purpose of this work, the images are up-sampled to 224×224 pixels to match the input specifications of the V-MoE patch tokeniser. A sample of its images is shown in Figure 5.2.

Office-Home The *Office-Home* dataset [24] is a widely used domain-adaptation benchmark containing approximately 15,500 images. It spans 65 object categories shared identically across four drastically different *visual domains*: *Art* (paintings and sketches), *Clipart* (clip-art collections), *Product* (catalogue images on white backgrounds), and *Real World* (standard photographs). A visualisation of the four domains is shown in Figure 5.3. Images from this dataset are processed at a 384×384 resolution.

Non-IID data heterogeneity In a standard centralised setting, data is assumed to be Independent and Identically Distributed (IID). In a cross-device federated topology, however, local datasets inherently reflect the specific usage patterns of individual clients, violating the IID assumption. Non-IIDness is not a single phenomenon: it manifests along several distinct axes —label priors, feature distributions, sample quantity, even temporal drift— and to widely varying degrees, from mild imbalance to nearly disjoint distributions, typically modulated by a concentration parameter. Rather than treating it generically, this work isolates two representative and complementary axes:

Label skew Intuitively, label skew describes clients that observe the *same kinds of inputs* but *different subsets of classes*. Formally [25], it occurs when clients share the same conditional appearance model, $P_i(x | y) \approx P_j(x | y)$, but differ sharply in their label priors, $P_i(y) \neq P_j(y)$. This simulates environments where a device is only exposed



Figure 5.2: CIFAR-100 image examples

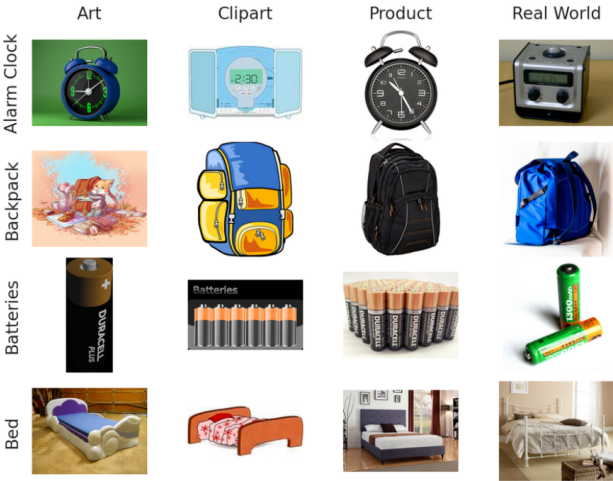


Figure 5.3: Office-Home domains example

5. Experimental Results

to a highly specific subset of categories, as originally demonstrated by Zhao et al. [26]. For instance, in a wildlife-monitoring federation a trail camera in a forest accumulates thousands of images of mammals and birds but essentially none of vehicles or furniture, while a client in an urban context observes the exact opposite.

Feature/Domain skew Intuitively, domain skew describes the opposite situation: clients observe the *same classes* but rendered in a *different visual style*. Following the taxonomy of non-IID data in federated networks [25], domain skew occurs when clients share an identical label space but draw inputs from mutually exclusive class-conditional feature distributions: $P_i(x | y) \neq P_j(x | y)$ while $P_i(y) = P_j(y)$. This captures scenarios where clients recognise the same objects under completely different conditions or styles. For example, three clients may all need to classify a chair, yet one observes photographic catalogue shots on white backgrounds, another hand-drawn clip-art, and a third artistic paintings —identical labels, drastically different pixel statistics.

Data partitioning strategies To induce the two non-IID regimes just defined in practice, custom partitioning pipelines were developed for the two datasets.

Domain-shift partitioning (Office-Home) Since each of the four visual domains shares the exact same 65 object categories, Office-Home acts as the canonical vehicle for inducing pure *domain skew*, also termed *domain shift*. Specifically, the federated environment is configured with 8 clients divided evenly into 4 clusters (yielding exactly two clients per cluster). Each cluster is exclusively assigned one of the four Office-Home visual domains. Within a given cluster, the data of that specific domain is uniformly divided between its two clients following an Independent and Identically Distributed (IID) split. As a result, both clients within the same cluster share an identical label prior (each retaining an equal share of the full 65-class support), whereas clients across different clusters experience completely disjoint marginal feature distributions. The resulting data allocation is visualised in Figure 5.4.

Pachinko-Thematic Partitioning (CIFAR-100) The term “Pachinko” originates from the hierarchical *Pachinko Allocation Model* used in topic modelling, inspired by the Japanese arcade game where balls cascade downwards through multiple layers of pegs. In this context, it describes a two-stage, hierarchical data distribution algorithm designed to induce severe but realistic *label skew*. Instead of a flat data split, this mechanism exploits the native two-level taxonomy of CIFAR-100 (where 100 fine labels are grouped into 20 coarse superclasses) to create heterogeneity on two simultaneous scales. Unlike the domain-shift configuration —where the four natural visual domains dictate four clusters— the two macro-themes defined here partition the pool of 8 clients into exactly two main clusters, each comprising four clients.

Stage 1: Thematic clusters First, the 20 coarse superclasses are divided into disjoint macro-themes: *Living Things* and *Things & Scenes*. Each cluster of clients is assigned one of these primary themes. To simulate realistic environmental variance, this allocation is governed by a theme concentration parameter c . In this configuration ($c = 0.85$), a cluster’s data pool targets approximately 85% in-theme samples and 15% out-of-theme images. This

induces severe *thematic missingness* between clusters: a client in a “Living Things” cluster will almost never observe images of vehicles or furniture.

Stage 2: Dirichlet skew Second, the data pooled within a cluster is distributed among its four clients. Rather than an equal split, a *Dirichlet distribution* with parameter $\alpha = 1.5$ is applied to the fine labels. Consequently, while clients within the same cluster share the identical theme, they exhibit distinctly non-uniform label priors. For example, within the “Living Things” cluster, one client might predominantly observe dolphins and fish, while another emphasises foxes and reptiles.

The resulting two-stage allocation is visualised in Figure 5.5.

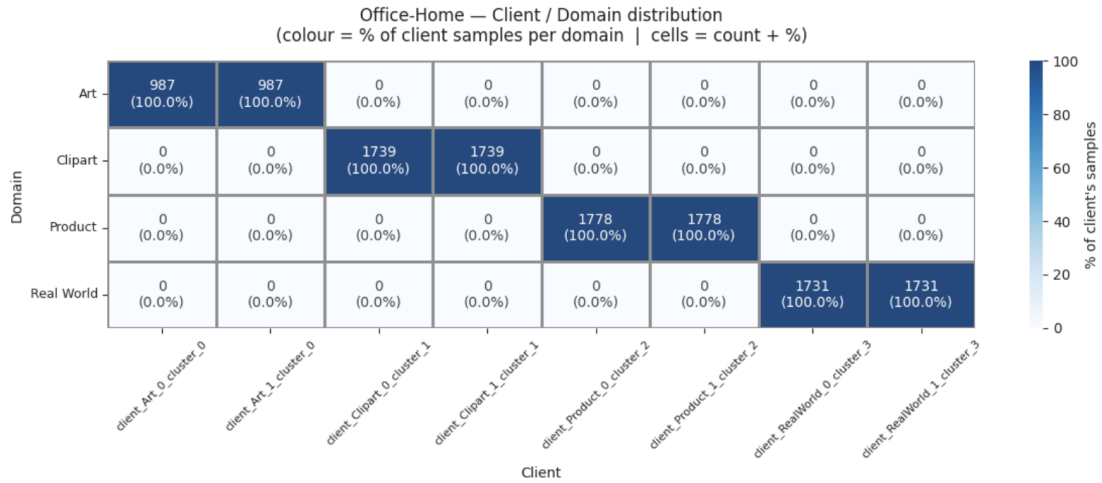


Figure 5.4: Domain shift partitioning on Office-Home

5.2 Evaluation Baselines

Before diving into the core experiments of this project, we first established a set of baseline models. We did this for two main reasons: first, to ensure our specific architecture and datasets align with the behaviors already well-documented in existing research; and second, to create a clear, reliable benchmark against which all our federated results can be measured. Evaluating a partial-replica federated V-MoE can be complex, as we need to separate two distinct factors. On the one hand, we have to account for any drop in accuracy caused purely by considering a federated setup (when compared to standard centralized training). On the other hand, we need to weigh the actual impact of using a sparse Mixture-of-Experts design instead of a traditional dense network. To keep these variables completely independent, we introduce our three baseline models, outlining their scopes and intended purposes, before ultimately presenting their empirical results and significance.

5.2.1 Centralised V-MoE Upper Bound

The first baseline establishes the absolute performance ceiling of the architecture. We reuse the exact fine-tuning recipe described in Section 4.5, but transpose it to a

5. Experimental Results

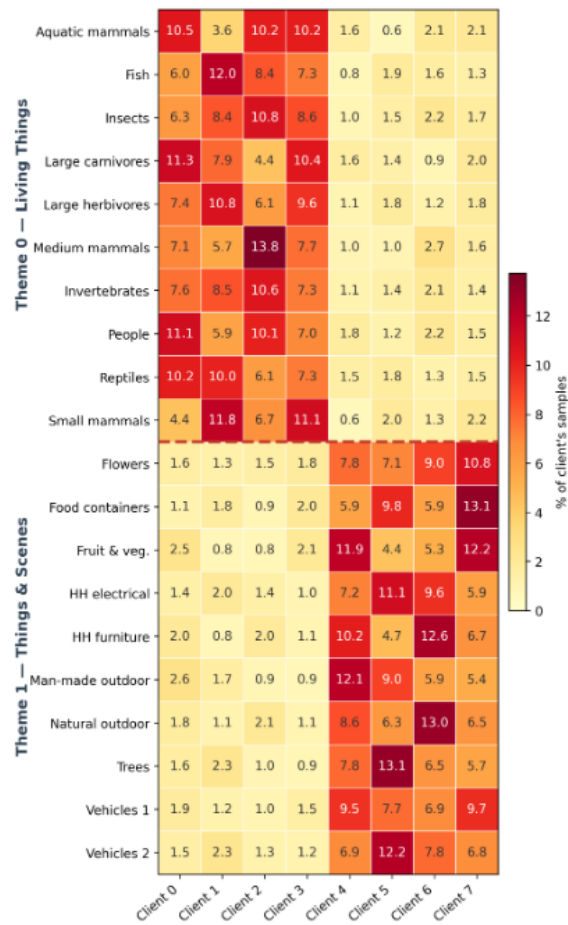


Figure 5.5: Pachinko-thematic data partitioning

conventional centralised (data-centre) setting, where all data is pooled on a single node and the entire expert pool is available to every token. The model retains its usual top- k gating (with $k = 2$) coupled with the conventional load-balancing auxiliary loss, and is trained for the same compute budget as the federated runs (50 centralised epochs mirroring the 50 communication rounds).

Being non-federated, this configuration removes every source of statistical degradation intrinsic to distributed learning, such as communication bottlenecks or weight dilution induced by FedAvg. Furthermore, to ensure a strict, one-to-one comparison between the centralized and federated paradigms, we deliberately adopted an IID data partition. This choice intentionally sets aside the additional complexities of non-IIDness, allowing us to isolate and evaluate the pure impact of the federated architecture itself. It is therefore not a competing method, but it provides the “ceiling” accuracy: a reference for the maximum model quality the backbone can attain on each target domain. Every federated configuration is expected to sit below this bound.

5.2.2 Full-Replica Federated V-MoE

Federated Learning inherently introduces a statistical penalty. To measure it in isolation, the second baseline trains the very same V-MoE-S/32 in a *full-replica* federated regime, where every client holds the complete expert pool, for 50 communication rounds on the same datasets used above.

Because this setup keeps the federated variable active while eliminating the partial-replica one, the gap between this baseline and the centralised upper bound quantifies the “federated penalty” alone. It thus serves as the primary internal reference against which the proposed partial-replica and dynamic-routing strategies are evaluated. The full-replica baseline is expected to perform slightly worse than the centralised upper bound, but it should still outperform any partial-replica configuration, which further restricts the model’s representational capacity.

5.2.3 Dense Federated ViT-S/32

The goal of this final baseline is to isolate the true advantage of the Mixture-of-Experts paradigm. To do this, we train a standard dense Vision Transformer (ViT-S/32) under the exact same federated setup and fine-tuning recipe as the MoE model.

We chose the dense ViT-S/32 to guarantee a computationally fair comparison. Because our MoE architecture uses top-2 routing, the active parameter count per token—and thus the cost of the forward and backward passes—remains very close to that of the dense ViT-S/32. The MoE model exceeds this cost by only a slight margin, which we quantify in Table 5.2.

Testing both models in the same federated environment lets us answer a clear question: given a roughly equal active compute budget on an edge device, does the sparse capacity of an expert pool actually outperform a purely dense network? Ultimately, this baseline highlights the practical trade-offs of the MoE design and provides the empirical evidence needed to justify the extra complexity of routing experts in a distributed system.

5.2.4 Results

Table 5.1 collects the outcome of the three reference runs on both benchmarks.

5. Experimental Results

Baseline	CIFAR-100			Office-Home		
	Acc@1 $\uparrow$	Acc@5 $\uparrow$	Val. Loss $\downarrow$	Acc@1 $\uparrow$	Acc@5 $\uparrow$	Val. Loss $\downarrow$
Centralised V-MoE	86.41%	98.61%	1.198	83.19%	96.50%	1.205
Full-Replica Federated V-MoE	85.69%	98.50%	1.221	82.87%	95.86%	1.287
Dense Federated ViT-S/32	83.72%	98.1%	1.280	80.85%	95.22%	1.365

Table 5.1: **Baseline results on the two benchmarks.** Accuracies are reported in % ($\uparrow$: higher is better); the validation loss is the final server-side cross-entropy ($\downarrow$: lower is better). All runs share an equivalent compute budget (50 centralised epochs $\equiv$ 50 communication rounds) and, except for the dense ViT-S/32, the V-MoE-S/32 backbone. The federated baselines use an IID data partition to isolate the federated penalty from the effect of non-IIDness. The three runs follow the expected ordering—Centralised $>$ Full-Replica $>$ Dense ViT-S/32: the federated penalty stays below 0.75 pp of Acc@1 on both benchmarks, whereas the sparse V-MoE surpasses the compute-matched dense ViT-S/32 by roughly 2 pp.

The three baselines follow the expected ordering on both benchmarks (Centralised V-MoE $>$ Full-Replica Federated V-MoE $>$ Dense Federated ViT-S/32) confirming the behaviour anticipated from the literature.

First, there is a *federated penalty*, but it is not high: only 0.72 percentage points (pp) of Acc@1 on CIFAR-100 (86.41% $\rightarrow$ 85.69%) and 0.32 pp on Office-Home (83.19% $\rightarrow$ 82.87%), with a correspondingly modest rise in validation loss. Under full participation, an IID partition, and SaMA aggregation, the federated setting recovers almost all of the centralised model quality on both benchmarks, the Acc@1 penalty staying well below one percentage point in either case.

Second, the *architectural advantage* of the expert design, isolated by comparing the two models under identical federated constraints, is clear and points the other way: the V-MoE outperforms the dense ViT-S/32 by 1.97 pp of Acc@1 on CIFAR-100 (83.72% $\rightarrow$ 85.69%) and 2.02 pp on Office-Home (80.85% $\rightarrow$ 82.87%), again at a lower validation loss.

Taken together, these baselines validate the pipeline against known results and fix the two reference points against which the partial-replica strategies of the following sections are assessed.

Capacity versus active compute A fair reading of the dense-versus-MoE comparison requires quantifying not only accuracy but also the computational terms on which it is obtained. Table 5.2 reports these terms as empirically measured on the two architectures.

The two models are not strictly iso-compute: because of top-2 routing, the V-MoE-S/32 activates $1.17\times$ the parameters per token and executes $1.14\times$ the multiply-accumulate operations (MACs) of the dense ViT-S/32. This modest overhead is, however, accompanied by a disproportionately larger representational budget, as the V-MoE stores $2.10\times$ the parameters of the dense model. The same asymmetry manifests in memory: sparse activation inflates the peak forward+backward VRAM by only $1.22\times$, despite the $2.08\times$ larger static weight footprint. This is precisely the promise of conditional computation—decoupling a model’s total capacity from its per-token cost—a property that a dense architecture, which activates every parameter for every token, cannot offer.

Therefore, we can conclude that while the V-MoE model demands higher static memory compared to the dense one, the use of conditional computation allows it to maintain a low per-token computational cost, roughly equal to the dense equivalent. Combined with a

Metric	Dense ViT-S/32	V-MoE-S/32	Ratio
Total parameters (capacity)	26.88 M	56.54 M	2.10×
Active parameters / token	26.88 M	31.34 M	1.17×
Sparsity (active / total)	100%	55.4%	—
MACs / image	1.356 G	1.550 G	1.14×
Static VRAM (weights)	0.10 GB	0.21 GB	2.08×
Peak VRAM (fwd+bwd)	0.60 GB	0.74 GB	1.22×

Table 5.2: **Capacity and active-compute comparison between the dense ViT-S/32 and the V-MoE-S/32.** “Active” quantities count only the parameters actually executed per token under top-2 routing; the MACs are reported as the top-2 dense-equivalent (i.e. the architectural compute cost, without token dropping). This is an architectural comparison: the peak VRAM is measured for a forward+backward pass *without* optimiser state (batch 32, FP32), and therefore differs by design from the full training-step figures of Table 4.2. The figures quantify conditional computation: the V-MoE stores 2.10× the parameters of the dense model, yet top-2 routing keeps its per-token cost close to the dense baseline (1.17× active parameters, 1.14× MACs) and inflates the forward+backward peak VRAM by only 1.22×.

significantly broader representational capacity, this explains why the research community has shifted towards MoE architectures in recent years.

5.3 Partial-Replica: Static and Dynamic

With the baselines in place, we now turn to the core subject of the campaign: the partial-replica V-MoE. This section fixes the premises shared by the static and dynamic analyses that follow—the evaluation scenario, the local expert budgets, the replication strategies used at initialisation, and the outcomes expected on both the accuracy and the system axes.

5.3.1 Premises and Expectations

Before presenting the empirical results, we briefly recall the scenario under evaluation and fix the accuracy expectations against which every configuration is read. In the partial-replica regime, each of the $N = 8$ clients no longer hosts the complete expert pool but only a local subset of $E_{\text{loc}} < E_{\text{tot}}$ experts per MoE layer. Whenever a client’s router selects an expert that is absent from its local allocation (a *ghost expert*), the *skip-token* mechanism bypasses the corresponding feed-forward computation, and that token gets zero-contribution from that layer.

Crucially, unlike the IID baselines of Section 5.2, all experiments are now conducted under the two non-IID partitions introduced previously in Section 5.1.3: the Pachinko thematic label-skew on CIFAR-100 and the domain-shift on Office-Home. The interaction between the routing preferences and the skewed local data is therefore expected to concentrate token demand on a restricted subset of experts, amplifying the impact of any locally missing module.

5. Experimental Results

Local expert pool Across all the experiments that follow, we investigate two constrained budgets for the local pool, $E_{\text{loc}} = 3$ and $E_{\text{loc}} = 4$, out of the $E_{\text{tot}} = 8$ experts populating each MoE layer. The case $E_{\text{loc}} = E_{\text{tot}} = 8$ would simply revert to the full-replica baseline of Section 5.2. Contrasting the two budgets is what allows us to isolate how the size of E_{loc} trades off against the final model accuracy.

5.3.2 Replication Strategies at Initialisation

Furthermore, we investigate two distinct strategies for replicating experts among the clients at initialisation. These strategies dictate the *replication degree*, defined as the number of clients that own a specific expert:

- **Balanced replication:** This strategy enforces a perfectly uniform distribution, meaning that each expert has the same number of replicas distributed across different clients. While the specific expert-to-client placement is randomised, the replication count per expert remains fixed by construction. Since each of the $N = 8$ clients holds E_{loc} experts, the federation contains $N \cdot E_{\text{loc}}$ expert slots in total; spreading them evenly over the $E_{\text{tot}} = 8$ experts yields a replication degree of $N \cdot E_{\text{loc}} / E_{\text{tot}} = E_{\text{loc}}$. It is therefore guaranteed to be exactly 3 in the $E_{\text{loc}} = 3$ scenario, and exactly 4 in the $E_{\text{loc}} = 4$ scenario.

By fixing the replication degree, this construction removes the variance in *expert coverage*, isolating the effect of the local pool size E_{loc} from any imbalance in how widely experts are shared across the federation.

- **Random (Unbalanced) replication:** On the other hand, this strategy intentionally creates an uneven distribution using a simple two-step process. First, it ensures that every expert is replicated on at least one client (this prevents the creation of “orphan” experts that would otherwise never be trained). Second, it fills the clients’ remaining empty slots randomly, making sure a client does not receive duplicates of the same expert. Because of this randomness, the number of owners per expert varies widely: some experts might be heavily shared across the federation (e.g., replicated on up to 7 clients), while others are held by very few (e.g., 1 or 2 clients).

This difference is fundamental for testing the system’s robustness. A widely shared expert receives stable updates because its weights are averaged across many clients. In contrast, an expert held by only a few clients relies on very limited data, making its learning highly sensitive to the specific (non-IID) data of those few users.

A final consideration applies to both strategies. Fixing the replication degree (balanced) or the coverage pattern (random) still leaves open *which* specific experts are mapped onto *which* client. Under the non-IID partitions, a pre-specialised expert is far more valuable on a client whose local domain routes to it than on one that never selects it, so a single arbitrary mapping could bias the outcome. To prevent any individual placement from driving the results, every configuration —balanced and random alike— is repeated over 3 independent placement seeds, and all reported figures are given as the mean and standard deviation across these runs. For the random strategy this average additionally absorbs the variability in coverage, whereas for the balanced one it isolates the placement effect alone.

5.3.3 Expected Outcomes

Two references bracket the partial-replica configurations. The full-replica federated V-MoE of Section 5.2 (Table 5.1: 85.69% on CIFAR-100, 82.87% on Office-Home), trained under an IID partition, sets the absolute ceiling. The more direct reference, however, is a full-replica run trained under the *same* non-IID partitions as the partial configurations: comparing against it isolates the pure cost of the reduced expert budget from the effect of the heterogeneity. Restricting each client to $E_{\text{loc}} < E_{\text{tot}}$ experts strictly removes representational capacity and forces a fraction of tokens to be skipped; no partial configuration is therefore expected to reach either reference, and the distance from the non-IID full replica is precisely the accuracy cost we set out to quantify.

However, since each client now hosts and updates only $E_{\text{loc}} < E_{\text{tot}}$ experts per MoE layer, we expect measurable reductions on the systems axis with respect to the full-replica baseline along two fronts. First, in *memory*: loading fewer expert modules should lower both the static VRAM (the resident weights) and the peak VRAM reached during the local forward and backward pass. Second, in *communication*: since a smaller number of expert (LoRA) parameters must be exchanged and aggregated at every round, the per-client payload transmitted to and from the server should shrink accordingly. In essence, the partial-replica regime is expected to trade a bounded loss in accuracy for a proportional relief of the very memory and bandwidth constraints that motivate it.

The dynamic case Finally, we explore the dynamic reassignment strategy with the explicit intent of weighing its advantages against its drawbacks over the static assignment. By periodically migrating experts toward the clients that most demand them, the mechanism seeks to reduce the reliance on ghost experts and to align each local pool more closely with the client’s data distribution.

5.4 Static Assignment

We now report the empirical behaviour of the *static* partial-replica regime, in which each client is permanently assigned its subset of E_{loc} experts before training and that never changes. The analysis is organised along three complementary axes: the model’s performance, the internal behaviour of the router, and the memory and communication cost.

5.4.1 Model Performance

Table 5.3 plots the global, server-side Top-1 and Top-5 accuracy of the four static configurations on both benchmarks. Because the partial-replica runs operate under non-IID data, each is compared against a full-replica reference trained on the identical non-IID partitions.

As expected, every partial configuration sits below the full-replica baseline and the distance from it is the *accuracy drop* of partial replication. Two regularities stand out.

More local experts help First, every configuration improves from $E_{\text{loc}} = 3$ to $E_{\text{loc}} = 4$, and the gain is larger for the random strategy (+0.93/+1.12 pp on CIFAR-100/Office-Home) than for the balanced one (+0.55/+0.18 pp), because the extra slot fills the coverage gaps that random replication may leave at $E_{\text{loc}} = 3$.

5. Experimental Results

Static Configuration	CIFAR-100		Office-Home	
	Acc@1 $\uparrow$	Acc@5 $\uparrow$	Acc@1 $\uparrow$	Acc@5 $\uparrow$
<i>Partial</i> $E_{\text{loc}} = 3$				
Balanced	83.75 $\pm$ 0.10 %	98.03 $\pm$ 0.12 %	80.64 $\pm$ 0.47 %	95.31 $\pm$ 0.16 %
Random	83.51 $\pm$ 0.48 %	98.00 $\pm$ 0.13 %	80.04 $\pm$ 0.86 %	95.03 $\pm$ 0.20 %
<i>Partial</i> $E_{\text{loc}} = 4$				
Balanced	84.30 $\pm$ 0.40 %	98.16 $\pm$ 0.11 %	80.82 $\pm$ 0.25 %	95.52 $\pm$ 0.22 %
Random	84.44 $\pm$ 0.49 %	98.17 $\pm$ 0.02 %	81.16 $\pm$ 0.71 %	95.46 $\pm$ 0.27 %
Full-replica	85.15%	98.29%	82.68%	95.82%

Table 5.3: **Static partial-replica accuracy** on the two benchmarks, for the balanced and random replication strategies at $E_{\text{loc}} \in \{3, 4\}$. Values are the mean $\pm$ standard deviation over 3 independent placement seeds ($\uparrow$: higher is better). Every partial configuration sits below the full-replica reference, yet the gap remains bounded—within about 1 pp on CIFAR-100 and 2 pp on Office-Home at $E_{\text{loc}} = 4$. Accuracy improves with the local budget, and the balanced strategy is the safer choice at the tight $E_{\text{loc}} = 3$ (higher mean, lower variance), whereas at $E_{\text{loc}} = 4$ the two strategies are statistically indistinguishable.

However, which strategy wins depends on the budget, and the driver is *variance*. The random replication is consistently the more variable—its Top-1 standard deviation exceeds the balanced one’s in every configuration (0.48 vs 0.10 on CIFAR-100, 0.86 vs 0.47 on Office-Home at $E_{\text{loc}} = 3$)—because an unlucky draw can assign a useful expert on very few clients, so that most of the federation skips its tokens; the balanced strategy avoids this by construction. Consequently, at the tight $E_{\text{loc}} = 3$ the balanced strategy is the safe choice, with a higher mean and far lower variance, whereas at $E_{\text{loc}} = 4$ the two are statistically indistinguishable (84.44% vs 84.30% on CIFAR-100; 81.16% vs 80.82% on Office-Home). The Top-5 accuracy stays near saturation ($\approx 98\%$ / $\approx 95\%$) regardless.

Bounded accuracy drop Second, the penalty is larger on Office-Home than on CIFAR-100 (about 1.5–1.9 pp vs 0.7–0.85 pp below the full replica at $E_{\text{loc}} = 4$); we account for this asymmetry in the routing analysis below. Despite this, the static partial replica preserves most of the model quality—at $E_{\text{loc}} = 4$ it stays within about one percentage point of the full replica on CIFAR-100 and two on Office-Home—confirming that a client can be restricted to half, or fewer, of the expert pool at a bounded accuracy cost.

5.4.2 Expert Specialisation

Figure 5.6 reports the *server-side* token routing of the final global model (round 50) on CIFAR-100, aggregated over the 20 coarse superclasses. Each cell gives the percentage of a category’s tokens dispatched to a given expert, shown separately for the two MoE layers (only the top-1 routing choice is considered). A preliminary observation is that these maps are, in practice, very similar across all four static configurations (balanced and random, at $E_{\text{loc}} \in \{3, 4\}$): the partial-replica topology does not reshape the global routing, which is instead dictated by the *strong prior* inherited from pre-training. Since the routers are globally shared and aggregated via FedAvg, we therefore report a single representative set.

MoE layer 0 specialisation The first MoE block is highly specialised: each category concentrates its tokens on a distinct subset of experts—for instance *trees*→E1 (57%),

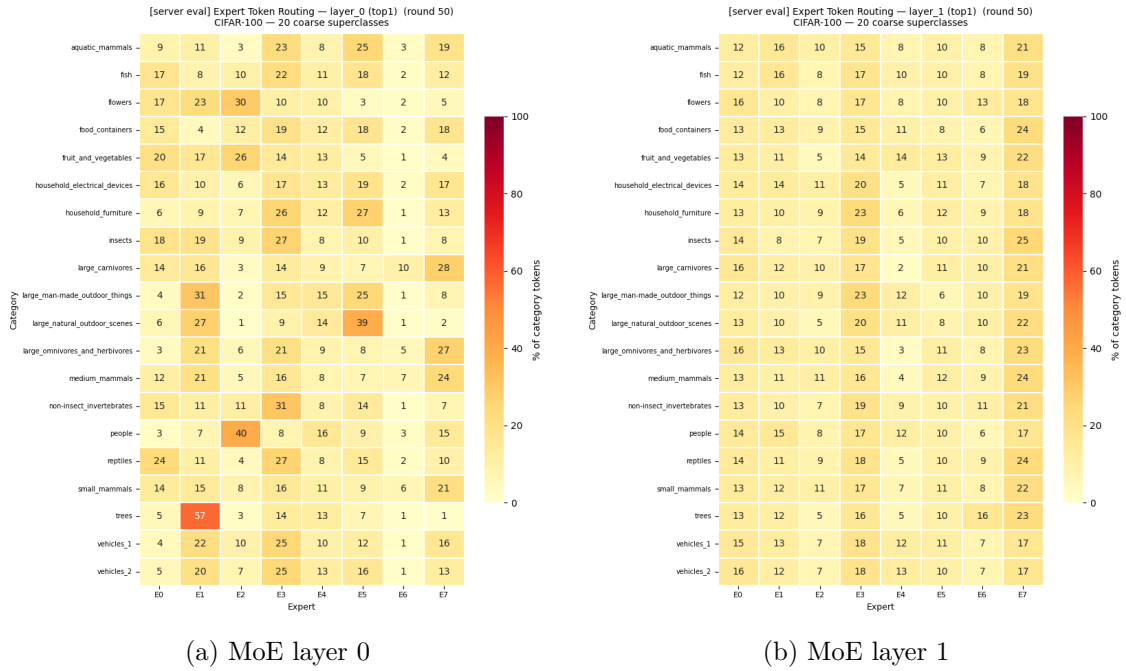


Figure 5.6: **Server-side top-1 expert token routing on CIFAR-100** at round 50, aggregated over the 20 coarse superclasses (each cell: the percentage of a category’s tokens sent to an expert). MoE layer 0 is strongly and semantically specialised—each category concentrates its tokens on a distinct subset of experts—whereas layer 1 routes almost uniformly and independently of the category, consistent with the representation homogenisation of deep ViT layers.

5. Experimental Results

people→E2 (40%) and *large natural outdoor scenes*→E5 (39%). This confirms that the experts inherit a marked specialisation from the ImageNet-1k pre-training and that, since CIFAR-100 is semantically close to ImageNet, this prior transfers easily.

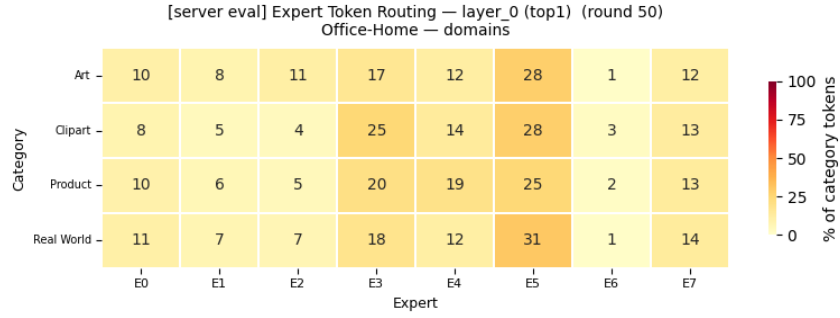
MoE layer 1 generalisation On the other hand, in the last MoE block the rows are near-identical: routing is close to uniform (except for a mild preference for E7) and essentially *independent* of the category. A likely cause is the homogenisation of token representations in deep ViT layers (*oversmoothing*): as self-attention pushes patches toward a common subspace, their inter-token variance shrinks and the router can no longer separate them, so the top-1 choice spreads out evenly.

Layer-wise asymmetry This layer-wise asymmetry has a direct consequence for the static regime. Because semantic specialisation is concentrated in layer 0, the penalty of a missing (ghost) expert is dominated by that layer: a client whose local categories map onto a specialised layer-0 expert it does not host will skip a large fraction of those tokens, whereas layer 1 —routing almost independently of the category— is largely robust to which experts are locally available. This localisation of the damage is precisely what the dynamic reassignment strategy studied later attempts to exploit.

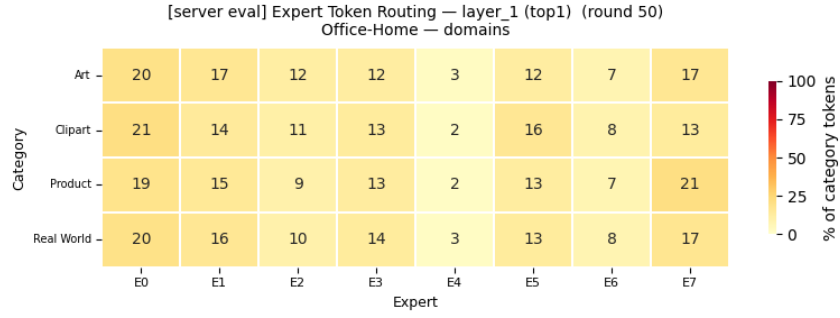
Office-Home analysis The same analysis on Office-Home shows a complementary scenario. Since the four Office-Home clusters share an identical label space and differ only in visual domain, Figure 5.7 aggregates the routing by *domain* rather than by category: each row is the expert usage of one domain (Art, Clipart, Product, Real World), averaged over all 65 classes. Two features are noteworthy.

Domain-invariance First, the rows are near-identical across the four domains (an image is routed the same way whether painted, drawn or photographed) so routing is *largely invariant to the visual domain*. Second, usage concentrates on a handful of experts (E5 dominates layer 0 at 25–31%, E0 dominates layer 1), far more than on CIFAR-100. The natural reading is that the router keys on the object’s *semantic content* rather than its rendering style, consistent with the object-level prior inherited from the ImageNet-1k pre-training; the concentration, in turn, reflects the narrower semantic span of Office-Home (all classes are household and office items) which occupies a smaller region of the expert space than the broad CIFAR-100 taxonomy.

Accuracy asymmetry Most importantly, this structure accounts for the accuracy asymmetry of Table 5.3. On Office-Home the demand is both *concentrated* (a few experts carry most tokens) and *homogeneous across clients* (domain-invariance means every cluster wants the *same* experts). Since under partial replication each expert lives on only three or four of the eight clients, the most-demanded expert is missing from several of them, which skip a large share of their tokens —hence the severe penalty. On CIFAR-100 demand is instead *distributed* and *heterogeneous across clusters*, so missing any single expert has less impact. The same concentration also makes the random strategy more volatile on Office-Home (Top-1 standard deviation 0.7–0.9 vs ≈ 0.5 on CIFAR-100): when everything hinges on one or two experts, an unlucky draw that under-replicates them swings the accuracy much more.



(a) MoE layer 0



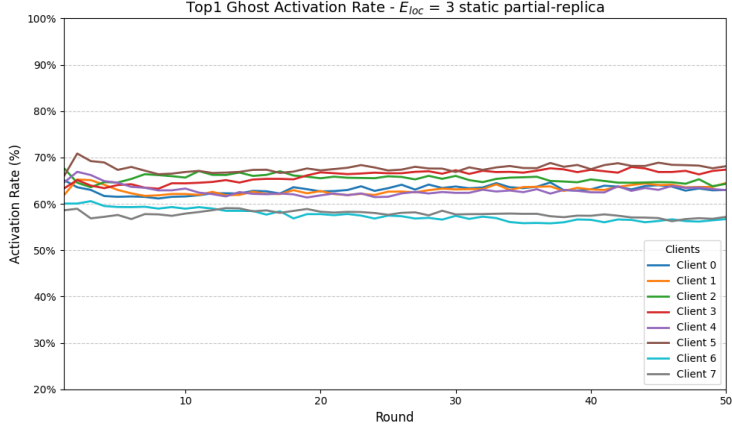
(b) MoE layer 1

Figure 5.7: **Server-side top-1 expert token routing on Office-Home** at round 50, aggregated over the four visual domains (each cell: the percentage of a domain’s tokens sent to an expert). Routing is largely invariant to the visual domain (the rows are near-identical whether the object is painted, drawn or photographed) and concentrates on a few experts (E5 dominates layer 0, E0 layer 1); this concentrated, cross-client-homogeneous demand accounts for the larger partial-replica penalty observed on Office-Home.

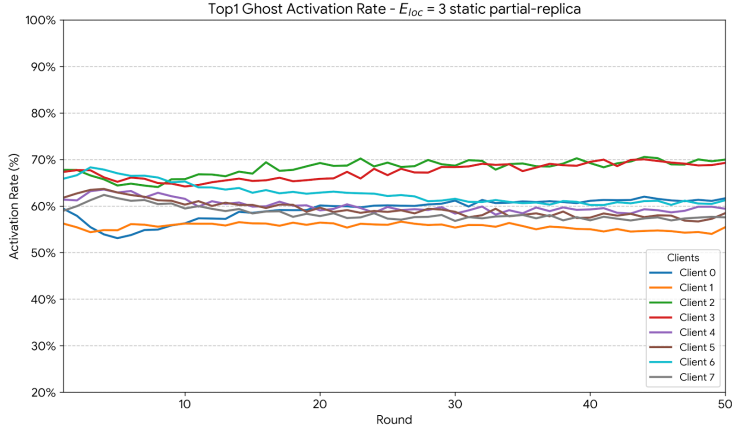
5. Experimental Results

5.4.3 Ghost Activations

To quantify the router’s behaviour under the partial-replica constraint, we track the *ghost top-1 rate*: the fraction of tokens whose highest-probability (top-1) routing choice falls on a ghost expert. Figures 5.8 and 5.9 report its evolution over the 50 communication rounds, one curve per client, for $E_{\text{loc}} = 3$ and $E_{\text{loc}} = 4$ respectively, on both benchmarks. Crucially, these plots refer to the *balanced* initial assignment; as established previously, its limited cross-seed variance provides a clearer, more representative baseline of the router’s structural behaviour compared to the volatile random strategy.



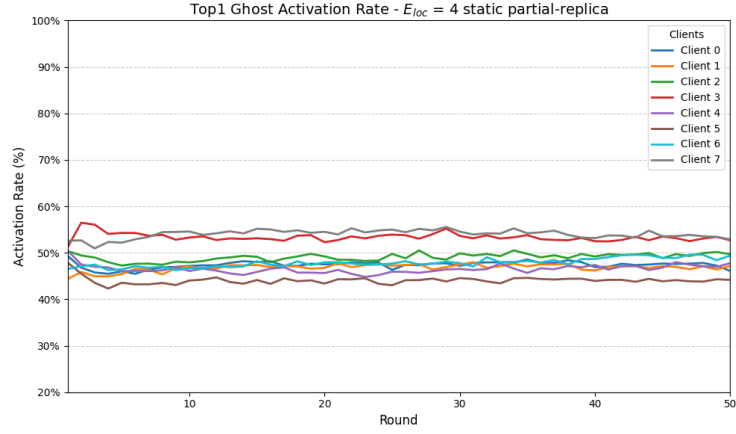
(a) CIFAR-100



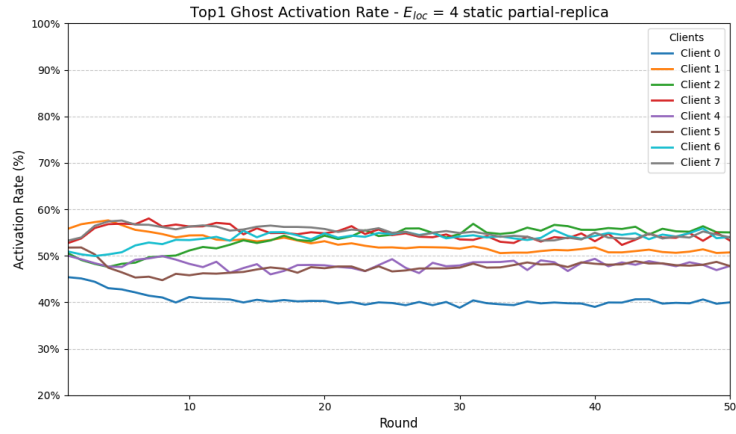
(b) Office-Home

Figure 5.8: **Per-client top-1 ghost activation rate for the $E_{\text{loc}} = 3$ static partial-replica** over the 50 communication rounds and under the balanced initial assignment, on CIFAR-100 (a) and Office-Home (b). The ghost rate centres around $\sim 62\%$ —closely tracking the structural share of missing experts $(E_{\text{tot}} - E_{\text{loc}})/E_{\text{tot}}$ —and stays essentially flat across rounds, evidencing the strong router’s prior coming from pre-training.

Determinism of the ghost rate The ghost top-1 rate is governed primarily by the local budget E_{loc} : it centres around $\sim 62\%$ for $E_{\text{loc}} = 3$ and drops to $\sim 50\%$ for $E_{\text{loc}} = 4$, closely tracking the structural baseline of missing experts $(E_{\text{tot}} - E_{\text{loc}})/E_{\text{tot}}$. This gives a concrete, mechanistic explanation for the accuracy gains obtained when scaling from three to four local experts (Table 5.3): the extra expert mechanically removes about $1/8$ (~ 12



(a) CIFAR-100



(b) Office-Home

Figure 5.9: **Per-client top-1 ghost activation rate for the $E_{\text{loc}} = 4$ static partial-replica** over the 50 communication rounds and under the balanced initial assignment, on CIFAR-100 (a) and Office-Home (b). The ghost rate centres around $\sim 50\%$ —about one eighth (~ 12 pp) below the $E_{\text{loc}} = 3$ case, matching the one extra local expert—and likewise remains stable throughout training, confirming the router’s rigidity regardless of the local budget.

5. Experimental Results

percentage points) of the top-1 requests that would otherwise land on ghost experts.

Router rigidity and the pre-trained prior The ghost rates remain remarkably stable throughout the fine-tuning, showing that the router does not learn to avoid the absent experts: the gating network keeps requesting its preferred semantic experts regardless of their local availability. Bound by a strong pre-trained prior and a minimal learning rate (10^{-5}) to prevent catastrophic forgetting, the gating network rigidly requests its preferred experts regardless of local availability. This behaviour further motivates a dynamic allocation strategy: since the router will not adapt to the local infrastructure, the infrastructure must adapt to the router.

Resilience through routing redundancy Despite these high ghost rates ($\sim 50\text{--}62\%$), the resulting accuracy loss is contained (1–2 pp). A top-1 ghost assignment does not always imply a dropped token: thanks to top-2 routing and the residual connections, a token whose top-1 expert is absent can still be processed by a locally present top-2 expert, or bypass the MoE layer unaltered through the residual path. We discuss the broader implications of this resilience in the conclusion of this section.

5.4.4 System Metrics

We now focus on the primary motivation and expected benefits behind this design: *system efficiency*. As anticipated, the true advantage of restricting the local expert pool lies in alleviating the hardware constraints and bandwidth concerns typical of cross-device federated networks. By hosting only a subset of experts, the client significantly reduces both the static memory (VRAM) required to load the model and the communication payload exchanged with the central server during each training round. We now quantify this fundamental trade-off, demonstrating how the static partial assignment trades a marginal loss in accuracy for predictable gains in memory and bandwidth footprint.

Communication payload Table 5.4 reports the per-client communication payload (in terms of parameters’ count) for the static partial-replica configurations, along with the percentage reduction relative to the full-replica baseline. The savings are substantial: at $E_{\text{loc}} = 4$ the client transmits 22.67% fewer parameters, and at $E_{\text{loc}} = 3$ the reduction reaches 28.34%. The table also includes a projection for a more aggressive MoE placement strategy (Every-2 V-MoE), which would further amplify the communication savings to 32.65% and 40.81% for $E_{\text{loc}} = 4$ and $E_{\text{loc}} = 3$, respectively. Despite paying the cost of a marginal accuracy drop, the partial-replica design therefore offers a clear benefit in terms of communication efficiency.

Memory and Computational Footprint Table 5.5 further expands the analysis by detailing the *on-device memory* and *computational footprint*.

The most immediate impact of the partial-replica strategy is observed in the model’s capacity (Part 1). By dropping the local expert count from the full $E_{\text{tot}} = 8$ to $E_{\text{loc}} = 4$, the static VRAM drops from 223.9 MB to 158.5 MB achieving a substantial reduction of nearly 29%. Restricting the pool further to $E_{\text{loc}} = 3$ shrinks the resident weights to 142.2 MB (a 36% reduction). Peak VRAM during the forward and backward passes decreases too, thanks to the *skip-token* mechanism. If every token found both of its top-2 experts locally, the peak would be identical in all three configurations. But with only 3 or 4 experts out of 8, a token whose top-1 or top-2 choice is a ghost expert skips that computation:

Configuration	Upload (Params) ↓	Download (Params) ↓	Bandwidth Savings ↑
Full-Replica	1.45 M	1.45 M	--
Partial $E_{\text{loc}} = 4$	1.12 M	1.12 M	22.67%
Partial $E_{\text{loc}} = 3$	1.04 M	1.04 M	28.34%
<i>Every-2 V-MoE</i>			
Full-Replica	2.03 M	2.03 M	--
Partial $E_{\text{loc}} = 4$	1.37 M	1.37 M	32.65%
Partial $E_{\text{loc}} = 3$	1.20 M	1.20 M	40.81%

Table 5.4: **Communication payload reduction under static partial-replica allocation.** “Upload” and “Download” denote the parameters exchanged by a single client in one communication round (the communication is symmetrical). The *Every-2 V-MoE Small* scenario projects the communication savings assuming an MoE layer is placed in every odd transformer block, given the same number of experts per MoE layer and number of transformer blocks. (↓: the lower the better, ↑: the higher the better). Restricting the local pool cuts the per-round payload by 22.67% at $E_{\text{loc}} = 4$ and 28.34% at $E_{\text{loc}} = 3$; under the denser Every-2 placement these savings would rise to 32.65% and 40.81%, respectively.

Part 1: Model Footprint

Configuration	Total Capacity (total params)	Trainable (LoRA & Full FT)	Static VRAM ↓ (weights)	Peak VRAM ↓ (fwd+bwd)
Full-Replica	57.64 M	1.45 M	223.9 MB	742.9 MB
Partial $E_{\text{loc}} = 4$	40.51 M	1.11 M	158.5 MB	700.5 MB
Partial $E_{\text{loc}} = 3$	36.22 M	1.03 M	142.2 MB	685.6 MB

Part 2: Per-Token Compute

Configuration	Active Params	MACs / Image ↓
Full-Replica	31.95 M	1.58 G
Partial $E_{\text{loc}} = 4$	31.95 M	1.37 G
Partial $E_{\text{loc}} = 3$	31.95 M	1.32 G

Table 5.5: **System profiling of the static partial-replica V-MoE architecture.** **Part 1** reports the resident cost of the model on a client. Shrinking the local pool from $E_{\text{tot}} = 8$ to $E_{\text{loc}} = 4$ and then to $E_{\text{loc}} = 3$ removes 8 and 10 of the 16 expert instances (two MoE layers $\times$ 8 experts), cutting static VRAM by 29% and 36% respectively. **Part 2** reports the per-image compute: the active parameter count stays pinned at the architectural ceiling of 31.95 M, since top-2 routing fixes the number of experts a token nominally invokes, whereas the realised MACs decline as the local pool shrinks. Peak VRAM and MACs both fall only because *skip-token* bypasses the computation of ghost experts, so no activations are stored for the tokens they would have served. (↓: the lower the better). Overall, the partial replica buys a third of the resident memory and 16% of the arithmetic back from an architecture in which the sparse component is, by design, a minority of the network.

5. Experimental Results

fewer activations are stored, so the model trades a little representational capacity for a lower memory (and compute) peak.

Crucially, Part 2 of Table 5.5 shows that the *active parameters* per token stay pinned at 31.95 M across all three configurations: because the conditional top-2 routing fixes how many experts a token nominally invokes, this figure is an *architectural ceiling* that is, by construction, independent of the pool size. The measured compute per image, however, has an opposite expected behaviour: it falls from 1.58 G MACs in the full-replica baseline to 1.37 G at $E_{\text{loc}} = 4$ and 1.32 G at $E_{\text{loc}} = 3$. As for the Peak VRAM metric, this is thanks to the skip-token mechanism.

5.4.5 The accuracy–system trade-off

The static partial-replica analysis delivers a coherent and, on balance, favourable verdict. Along the accuracy axis, restricting each client to half—or fewer—of the expert pool costs only 1 to 2 percentage points of Top-1 accuracy with respect to the full replica trained under the same non-IID partitions (Table 5.3). This key outcome relies on three reinforcing design factors:

- **Sparsity-aware Model Aggregation:** the SaMA scheme protects the experts from dilution, weighting each update by its actual token utilisation. Unlike standard FedAvg, a sparsely replicated expert is not averaged into irrelevance by clients that never hosted it nor routed any token to it.
- **Top-2 Routing and Residual Stream:** top-2 routing and the residual pathway provide a built-in redundancy: a token whose top-1 choice is a ghost expert can still be served by a locally present second choice, or pass through unaltered, which is why ghost rates of 50–62% translate into only single-digit accuracy losses.
- **Buonded Sparse Footprint:** the specific V-MoE-S/32 footprint—a modest pool of $E_{\text{tot}} = 8$ experts confined to just two *last-2* MoE layers—keeps the sparse component a bounded fraction of the network, so that dividing it perturbs the model only marginally.

Against this bounded cost, the system-level gains are concrete and solid. On the two metrics that matter most for cross-device deployment, the partial assignment delivers up to 28% savings in per-round communication payload and up to 36% in static VRAM (Tables 5.4 and 5.5), while ghost skipping additionally trims the compute actually executed on the device—easing the processor and battery load, an effect that grows as E_{loc} shrinks. The very architectural property that caps the accuracy penalty—few experts, few MoE layers—also caps the *absolute* magnitude of these savings; yet the *Every-2* projection shows that both fronts scale directly with MoE density, reaching a 41% payload reduction under a denser placement. In other words, the more an architecture leans on its experts, the more the partial-replica regime has to offer. The static assignment therefore establishes a clean, cost-isolated operating point: a marginal, well-characterised accuracy sacrifice bought back as memory and bandwidth head-room that would otherwise gate participation on constrained edge hardware.

From static rigidity to dynamic adaptation This favourable trade-off, however, is achieved *in spite of*, rather than *because of*, the router’s behaviour. Our ghost-activation analysis shows that the pre-trained gating network stubbornly requests its preferred experts regardless of local availability. This mismatch causes noticeable penalties, especially

when demand is highly concentrated and homogeneous across clients (like in the Office-Home setup, where every cluster wants the same few experts).

This observation motivates a natural next step: if the router will not adapt to the local infrastructure, then the infrastructure should adapt to the router. The following section explores this idea by introducing *dynamic expert reassignment*. This protocol periodically migrates experts to the clients that demand them most, and evaluates whether the resulting alignment justifies the added communication and stability costs.

5.5 Dynamic Reassignment

Following the latest analysis, we now focus on the dynamic expert reassignment strategy. This protocol, formalised in Chapter 4, addresses the main limitation of the static approach: the rigid allocation of experts to clients under non-IID conditions. Instead of pinning each client to a fixed expert subset for the entire run, the central server monitors the demand every client expresses for its ghost experts and, at periodic swap rounds, migrates experts towards the clients that most need them, one capacity-preserving swap at a time.

We report the results along the same axes used for the static case: the global model performance, the evolution of the router’s ghost demand, the changes in the clients’ local expert pools, and the cost of the migrations themselves. The final goal is to evaluate the concrete benefits of this topological adaptation, balancing them directly against the overhead caused by the expert transfers.

5.5.1 Model Performance

Table 5.6 reports both the dynamic and the static accuracy results for comparison. Crucially, the two paradigms share the *same three placement seeds*, so the comparison is strictly paired: any difference reflects the reassignment mechanism alone, and not a more (or less) fortunate set of random draws.

Dynamic matches static in the mean On the average accuracy, the dynamic protocol neither dramatically helps nor hurts. The mean Top-1 shifts by only a fraction of a point with respect to the matching static configuration (on CIFAR-100, +0.16 pp for $E_{\text{loc}} = 4$ balanced and -0.38 pp for random; on Office-Home, +0.20 and +0.11 pp), and the Top-5 stays at saturation. However, its contribution must be interpreted in the *stability* of the results.

Dynamic collapses the cross-seed variance This is exactly where the clearest effect emerges. At $E_{\text{loc}} = 4$, the cross-seed standard deviation shrinks sharply under the dynamic regime: on CIFAR-100 it falls from $\pm 0.40/\pm 0.49$ (balanced/random, static) down to $\pm 0.05/\pm 0.05$, and the volatile static random on Office-Home drops from ± 0.71 to ± 0.25 . Each configuration thus becomes highly reproducible, and the reduction is largest precisely for the *random* initialisation—the strategy that may create high disparity on the experts’ replication degree.

This confirms the mechanism functions as intended: by migrating experts towards the clients that demand them, the server steers every run towards a similar, demand-aligned final state, neutralising the dependence on the initial placement.

5. Experimental Results

Configuration	CIFAR-100		Office-Home	
	Acc@1 $\uparrow$	Acc@5 $\uparrow$	Acc@1 $\uparrow$	Acc@5 $\uparrow$
<i>Partial Static</i>				
$E_{\text{loc}} = 3$ Balanced	83.75 $\pm$ 0.10 %	98.03 $\pm$ 0.12 %	80.64 $\pm$ 0.47 %	95.31 $\pm$ 0.16 %
$E_{\text{loc}} = 3$ Random	83.51 $\pm$ 0.48 %	98.00 $\pm$ 0.13 %	80.04 $\pm$ 0.86 %	95.03 $\pm$ 0.20 %
<i>Partial Dynamic</i>				
$E_{\text{loc}} = 3$ Balanced	83.92 $\pm$ 0.42 %	98.05 $\pm$ 0.06 %	80.58 $\pm$ 0.51 %	95.36 $\pm$ 0.22 %
$E_{\text{loc}} = 3$ Random	83.65 $\pm$ 0.21 %	97.99 $\pm$ 0.05 %	79.98 $\pm$ 0.94 %	95.38 $\pm$ 0.03 %
<i>Partial Static</i>				
$E_{\text{loc}} = 4$ Balanced	84.30 $\pm$ 0.40 %	98.16 $\pm$ 0.11 %	80.82 $\pm$ 0.25 %	95.52 $\pm$ 0.22 %
$E_{\text{loc}} = 4$ Random	84.44 $\pm$ 0.49 %	98.17 $\pm$ 0.02 %	81.16 $\pm$ 0.71 %	95.46 $\pm$ 0.27 %
<i>Partial Dynamic</i>				
$E_{\text{loc}} = 4$ Balanced	84.46 $\pm$ 0.05 %	98.16 $\pm$ 0.05 %	81.02 $\pm$ 0.25 %	95.52 $\pm$ 0.18 %
$E_{\text{loc}} = 4$ Random	84.06 $\pm$ 0.05 %	98.11 $\pm$ 0.05 %	81.27 $\pm$ 0.25 %	95.52 $\pm$ 0.13 %
Full-replica	85.15%	98.29%	82.68%	95.82%

Table 5.6: **Dynamic and static partial-replica accuracy** on the two benchmarks, for the balanced and random initial-assignment strategies at $E_{\text{loc}} \in \{3, 4\}$. Values are the mean $\pm$ standard deviation over 3 independent placement seeds ($\uparrow$: higher is better). Per column, **bold** marks the best configuration within each local budget ($E_{\text{loc}} = 3$ or $E_{\text{loc}} = 4$) and **bold italic** the best overall (full-replica excluded), with ties broken by the lower standard deviation. The full-replica row is the non-IID reference reproduced from Table 5.3.

The budget must be loose enough This stabilising effect, however, is confined to $E_{\text{loc}} = 4$. At the tight $E_{\text{loc}} = 3$ the dynamic standard deviations remain comparable to their static counterparts. With only three local slots, the capacity-preserving swaps have too little room to satisfy every client’s demand at once. This is a limitation we analyse deeper further below.

5.5.2 Expert Specialisation

We first verify whether the reassignment reshapes the router’s token-to-expert heat-maps. Across the three placement seeds and both local budgets, the maps proved remarkably stable, differing only by fractions of a percentage point between them; we therefore show a single representative pair per benchmark. Figures 5.10 and 5.11 show the server-side top-1 routing under the dynamic topology, to be read against their static counterparts (Figures 5.6 and 5.7).

Routing is preserved The maps are essentially those of the static regime. On CIFAR-100 the layer-0 specialisation is intact—*trees* still concentrate on E1 ($\sim 60\%$) and *people* on E2 ($\sim 37\%$)—and layer 1 remains near-uniform; on Office-Home the demand keeps concentrating on a strict subset of experts, invariant to the visual domain. This is expected, since the routers are globally shared and FedAvg-aggregated regardless of the assignment strategy: reassignment adapts the *infrastructure* (which client hosts which expert) rather than the *router* (which expert a token asks for).

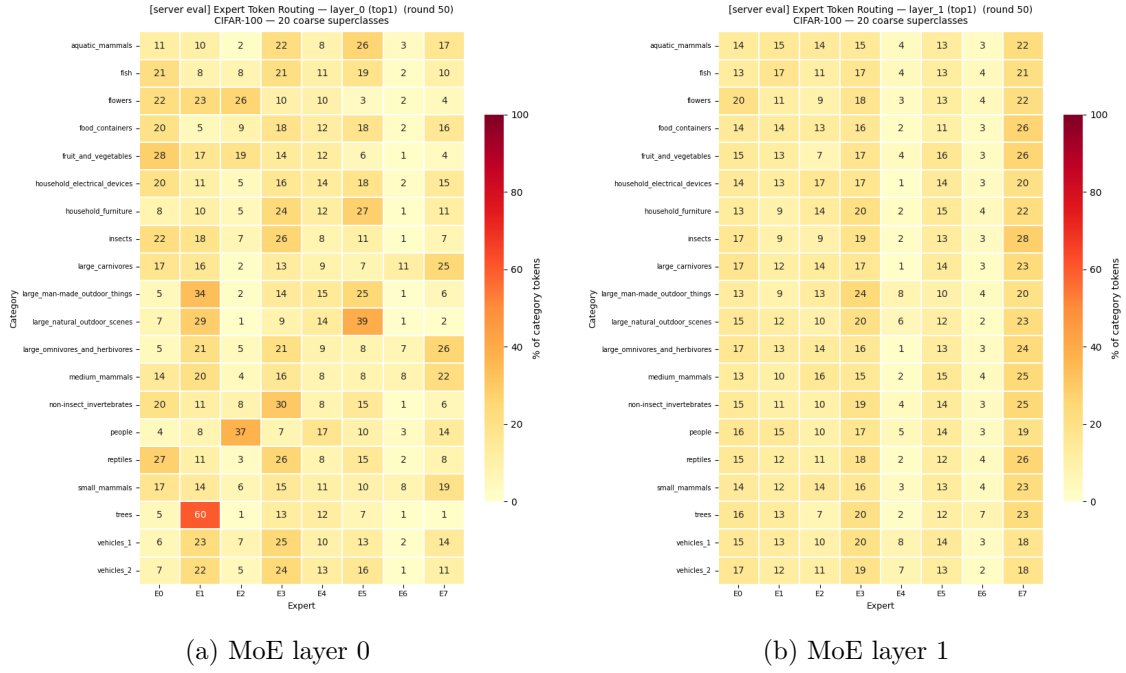


Figure 5.10: **Server-side top-1 expert token routing on CIFAR-100 under dynamic reassignment** at round 50, aggregated over the 20 coarse superclasses (each cell: the percentage of a category’s tokens sent to an expert). The maps are indistinguishable from the static case (Figure 5.6): layer 0 keeps its semantic specialisation and layer 1 stays near-uniform, confirming that reassignment leaves the globally shared router untouched.

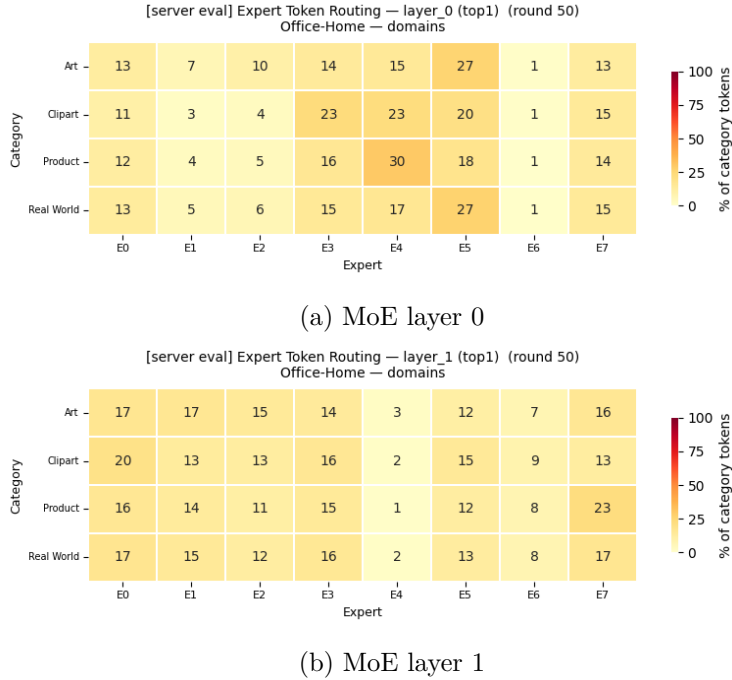


Figure 5.11: **Server-side top-1 expert token routing on Office-Home under dynamic reassignment** at round 50, aggregated over the four visual domains (each cell: the percentage of a domain’s tokens sent to an expert). Routing stays domain-invariant and concentrated as in the static case (Figure 5.7).

5.5.3 Ghost Activations

The accuracy behaviour above is best understood by looking directly at how well each local pool matches the router’s demand. We therefore reuse the *server-side ghost top-1 rate* of the static analysis (Figures 5.8 and 5.9), now tracked over the 50 rounds under the adaptive topology. Figure 5.12 reports it for both budgets and benchmarks, comparing the dynamic *balanced* and *random* runs (each averaged over the three placement seeds) against the *static* partial-replica reference.

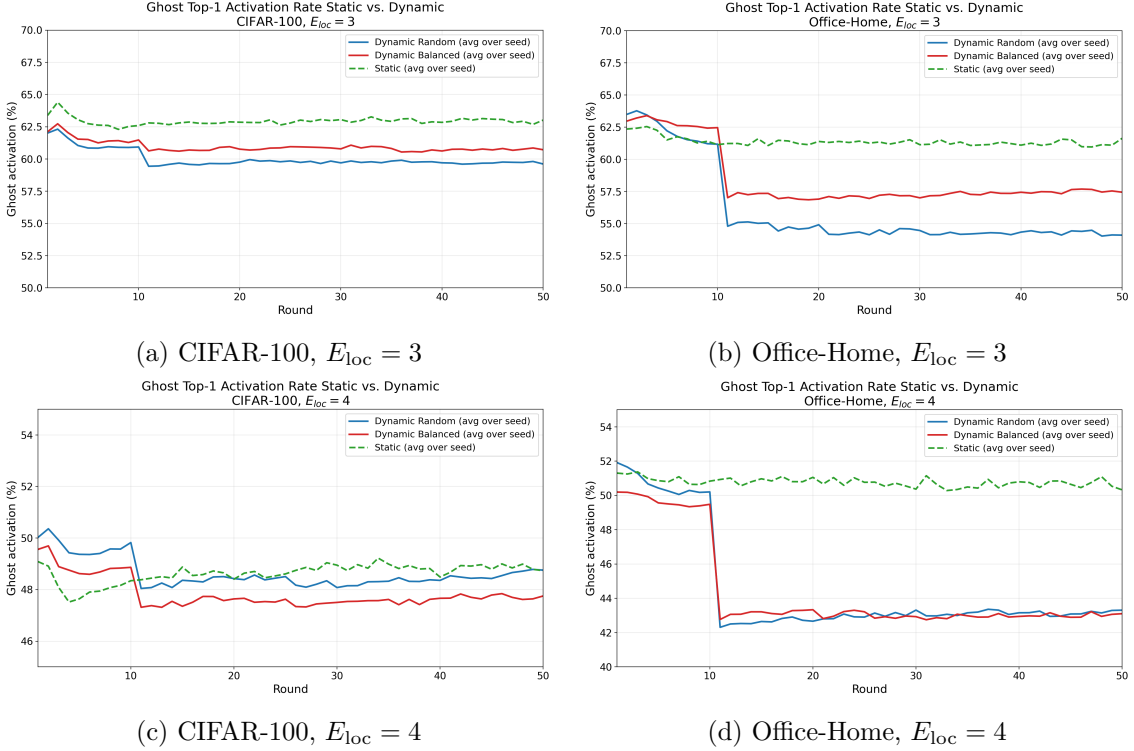


Figure 5.12: **Server-side top-1 ghost rate: static versus dynamic reassignment** over the 50 communication rounds, for both budgets and benchmarks. Each panel plots three curves: the dynamic *balanced* and *random* runs (each the mean over the three placement seeds) against the flat *static* partial-replica reference. Unlike the static curve—pinned at its structural baseline of $\sim 62\%$ ($E_{loc} = 3$) and $\sim 50\%$ ($E_{loc} = 4$)—the dynamic curves decline as experts migrate towards demand: sharply on Office-Home (~ 7 pp) and only mildly on CIFAR-100 (~ 2 pp); on the Office-Home $E_{loc} = 4$ case the two initialisations converge to the same final rate, thanks to the higher local budget.

The ghost rate now declines The static reference curve is flat—pinned at the structural baseline of $\sim 62\%$ ($E_{loc} = 3$) and $\sim 50\%$ ($E_{loc} = 4$)—because a fixed topology and a rigid router leave the share of misdirected requests unchanged. Under reassignment it instead *falls*: from $\sim 62\%$ to $\sim 60\%$ (CIFAR-100) and $\sim 55\%$ (Office-Home) at $E_{loc} = 3$, and from $\sim 50\%$ to $\sim 48\%$ (CIFAR-100) and $\sim 43\%$ (Office-Home) at $E_{loc} = 4$. The router still requests the same experts; it is the *infrastructure* that moves them closer to demand, exactly the inversion the previous section set out to test. Almost all of the reduction is realised at the first post warm-up swap round, after which the curve plateaus.

Alignment tracks the demand structure The reduction is largest on Office-Home (~ 7 pp against ~ 2 pp on CIFAR-100), where demand is concentrated and homogeneous across clients, so moving a few experts realigns many of them at once. The clearest case is Office-Home at $E_{\text{loc}} = 4$, where the balanced and random initialisations settle onto the *same* ghost rate ($\sim 43\%$). This is the topological echo of the accuracy stabilisation seen earlier at $E_{\text{loc}} = 4$ (Table 5.6): the mechanism absorbs the initial placement and drives every run to the same demand-aligned state.

5.5.4 Swaps Analysis

Having shown that reassignment aligns the topology with demand, we now analyze the migrations themselves: how many swaps occur, when they happen, and which clients are involved. We track these dynamics using per-round decision logs across both benchmarks. These logs perfectly match the actual changes in local expert pools, with zero discrepancies across all eight configurations. Table 5.7 summarizes these results, each averaged over its independent placement seeds.

Partial-Dynamic Configuration	Swaps (avg over seed)	Swap location layer 0 / layer 1	Rejected Swaps			Changed slots (round 1→50)
			Admission Margin	Max Replicas	Conflict	
<i>CIFAR-100</i>						
$E_{\text{loc}} = 3$ Balanced	4.0	8% / 92%	73.7	11.3	7.0	25%
$E_{\text{loc}} = 3$ Random	5.0	13% / 87%	72.3	9.3	9.3	31%
$E_{\text{loc}} = 4$ Balanced	7.3	23% / 77%	66.3	22.3	0.0	44%
$E_{\text{loc}} = 4$ Random	7.0	21% / 79%	61.5	27.5	0.0	41%
<i>Office-Home</i>						
$E_{\text{loc}} = 3$ Balanced	8.0	58% / 42%	68.3	8.3	11.3	50%
$E_{\text{loc}} = 3$ Random	9.7	62% / 38%	64.7	8.7	13.0	58%
$E_{\text{loc}} = 4$ Balanced	11.7	49% / 51%	56.7	26.3	1.3	63%
$E_{\text{loc}} = 4$ Random	11.3	50% / 50%	57.0	27.7	0.0	65%

Table 5.7: **Swap dynamics of the partial-dynamic expert reassignment on both benchmarks**, for the balanced and random initial placements at $E_{\text{loc}} \in \{3, 4\}$. Every value is an average per random seed. *Swaps* is how many expert exchanges actually took place (each one swaps a local expert for a wanted one, so the client always keeps the same number of experts). *Swap location* splits those swaps between the two MoE layers (the two figures are the percentage happening in layer 0 and in layer 1). The three *Rejected Swaps* columns count the exchanges that were proposed but refused, grouped by reason: *Admission Margin*, the wanted expert was not clearly better than the one it would replace; *Max Replicas*, that expert was already stored on the maximum number of clients allowed; *Conflict*, the request was dropped after several clients competed for the same move. The remaining swap constraints (minimum-demand threshold, cool-down, and a minimum-replicas floor) never triggered in these runs and are omitted. *Changed slots* is the share of client-layer slots that host a different expert at the last round than at the first. On both datasets the local budget is what matters most: giving each client one more expert ($E_{\text{loc}} : 3 \rightarrow 4$) produces more swaps and more changed slots, removes the conflict rejections and multiplies the “Max Replicas” ones. Office-Home also reorganises about twice as much as CIFAR-100, and moves more experts in layer 0.

When and how much Almost every swap happens at the first swap round (round 10), after which the topology settles—exactly what gives the ghost-rate curves of Figure 5.12 (long plateau). How much reorganisation takes place is set mainly by the local budget:

5. Experimental Results

going from $E_{\text{loc}} = 3$ to $E_{\text{loc}} = 4$ roughly doubles both the swaps and the changed slots. The dataset amplifies the volume: Office-Home reorganises about twice as much as CIFAR-100, while the initial placement matters far less.

Why the budget matters The most frequent reason a swap is rejected is the *admission margin*: the requested expert is not significantly better than the current one. This happens because most useful swaps occur early in the training. After this initial phase, clients already hold the experts they need most, so new candidates offer little improvement. Having a small pool of only $E_{\text{tot}} = 8$ experts per layer makes this even more common, leaving few alternatives once the best experts are taken.

The local budget decides the rest. Only when $E_{\text{loc}} = 4$ do *conflict* rejections drop to zero (down from ~ 7 –13 per seed). At this budget, swaps happen easily enough to fix the network’s topology in every run. This explains the accuracy results in Table 5.6: performance stabilizes (variance collapses) only at $E_{\text{loc}} = 4$. In contrast, the stricter $E_{\text{loc}} = 3$ budget leaves coverage gaps, making the final outcome heavily dependent on the initial expert setup.

High demand leads to high replication A final check confirms that swaps are meaningful: Table 5.8 compares how many tokens the server routes to each layer-0 expert versus how many clients actually host it by the end of training. We recall that the maximum number of replicas for any single expert is capped at 6 out of 8 clients.

The results show a strong correlation (Pearson $r = 0.95$). The most requested expert, E5 (24% of tokens), always hits the maximum limit of 6 replicas. On the other hand, the least requested expert, E6 (1%), stays at the minimum floor of 2 replicas. This proves the success of the $E_{\text{loc}} = 4$ setup: it gives the system enough room to replicate the most useful experts across most of the network. The system independently discovers the router’s preferences just by tracking demand.

Expert	E5	E4	E3	E7	E0	E2	E1	E6
Token demand (%)	24.0	19.3	18.4	13.5	10.9	7.2	5.5	1.2
Final replicas (/6)	6.0	4.3	5.3	4.5	4.3	3.0	2.5	2.0

Table 5.8: **Server-side demand versus client-side replication on Office-Home** (MoE layer 0, $E_{\text{loc}} = 4$), averaged over the six runs ($3 \text{ seeds} \times 2 \text{ initialisations}$), experts ordered by demand. “Token demand” is the share of tokens the global router sends to each expert; “Final replicas” is how many of the eight clients host it at the last round. The two independent measures correlate strongly: the reassignment replicates most the experts the router demands most—E5 at the ceiling (6/8), E6 at the floor (2/8)—confirming that it converges on a real specialisation hierarchy rather than churning at random.

5.5.5 The Cost of Dynamism

The reassignment protocol trades its topological alignment with communication. This subsection quantifies that price exactly, closing the loop on the system analysis of the dynamic case.

An ordinary round is symmetrical Under the PEFT regime a round moves *only* trainable tensors in both directions: the client uploads them (upload), the server aggregates them, and it sends them back (download). Three components define each one-way payload $P_{\text{one-way}}$: the parameters kept in full fine-tuning (e.g., `router.dense_linear`, `pos_embed`), the LoRA matrices of the shared frozen backbone, and the LoRA matrices of the E_{loc} experts the client owns in each of the L MoE layers. Grouping the first two into a single term w_{shared} :

$$P_{\text{one-way}} = \underbrace{w_{\text{shared}}}_{\text{full-FT + backbone LoRA}} + \underbrace{E_{\text{loc}} \cdot L \cdot w_{\text{LoRA}}}_{\text{owned experts}} \quad (5.1)$$

With $L = 2$ and the units measured in Table 5.9, $P_{\text{one-way}} = 1.04$ M parameters at $E_{\text{loc}} = 3$ and 1.12 M at $E_{\text{loc}} = 4$, per client, recovering the static figures of Table 5.4.

A swap round breaks the symmetry Reassignment violates that invariant. When a swap grants a client an expert it does not own, the server must transmit that expert *in full*: its frozen base weights alongside its aggregated LoRA matrices so far. The overhead is consequently *one-directional*. The upload is untouched, because a client always trains and returns exactly E_{loc} experts per layer, whatever their identity. On the other hand, the download payload grows by

$$P_{\text{swap}} = \underbrace{w_{\text{base}}}_{\text{frozen}} + \underbrace{w_{\text{LoRA}}}_{\text{trainable}} \quad (5.2)$$

for each incoming expert. A client may well receive more than one expert in the same reassignment round, as the protocol treats the L MoE layers independently and may involve one or all of them. Writing $s_c(t)$ for the number of experts entering client c in different MoE layers at round t , its download becomes:

$$P_{\text{one-way}} + s_c(t) \cdot P_{\text{swap}} \quad (5.3)$$

against an unchanged upload of $P_{\text{one-way}}$.

The true cost of a dynamic run Summing over the $R = 50$ rounds, and adding the ghost demand telemetry P_{tele} that each client piggybacks on every update, the overall communication cost of a dynamic run for client c is:

$$\text{Comm}_{\text{dyn}}(c) = \underbrace{2 R P_{\text{one-way}}}_{\text{static baseline}} + \underbrace{N_{\text{swaps}}(c) P_{\text{swap}}}_{\text{download only}} + \underbrace{R P_{\text{tele}}}_{\text{upload only}} \quad (5.4)$$

where $N_{\text{swaps}}(c) = \sum_t s_c(t)$ is the number of experts the client acquires over the whole run, and the factor 2 on the first term accounts for the symmetrical round trip established above. The entire price of dynamism is thus confined to the two rightmost terms, which the remainder of this subsection measures.

The experts' base weights are the whole bill Table 5.9 resolves the two terms of Equation 5.2 into measured quantities, and their imbalance is extreme: $w_{\text{base}} = 2.100$ M against $w_{\text{LoRA}} = 0.041$ M, a percentage ratio of 98.1% to 1.9%. A swap thus costs a client roughly two full rounds of download ($P_{\text{one-way}} = 1.04$ M params against $P_{\text{swap}} = 2.14$ M params). The telemetry term is instead negligible: 0.24 MB over the whole federation (8 clients) and the whole run (50 rounds).

5. Experimental Results

What is transmitted	Symbol	Params	FP32 size
<i>per-round, one-way</i>			
Full-FT modules + backbone LoRA	w_{shared}	0.790 M	3.01 MB
LoRA of one owned expert, one MoE layer	w_{LoRA}	0.041 M	0.16 MB
One-way payload, $E_{\text{loc}} = 3$	$P_{\text{one-way}}$	1.04 M	3.95 MB
One-way payload, $E_{\text{loc}} = 4$	$P_{\text{one-way}}$	1.12 M	4.27 MB
<i>On the download of a swap round only</i>			
Frozen base weights of one expert	w_{base}	2.100 M	8.01 MB
Aggregated LoRA of one expert	w_{LoRA}	0.041 M	0.16 MB
Swap payload	P_{swap}	2.141 M	8.17 MB
<i>In the dynamic upload case only</i>			
Ghost demand telemetry (all rounds, all clients)	P_{tele}	– –	0.24 MB

Table 5.9: **Measured payload units of the ordinary round and of one expert migration** on V-MoE-S/32, in FP32, for a single client. The upper block lists the terms of Equation 5.1, exchanged symmetrically at every round; the lower two list the terms that Equation 5.4 adds on top, and that exist only under dynamic reassignment. A swap must ship the incoming expert in full, because a partial-replica client holds no copy of an expert it does not own: its frozen base weights make up 98% of that transfer and its LoRA matrices only 2%, so the price of dynamism is the base weights and not the adapters. Migrating one expert costs its recipient roughly two rounds’ worth of download ($2.1\times$ at $E_{\text{loc}} = 3$, $1.9\times$ at $E_{\text{loc}} = 4$), whereas the telemetry driving the whole protocol is negligible: 0.24 MB accumulated over all 50 rounds and all 8 clients.

An extra cost that gets smaller over time Table 5.10 turns the swap counts of Table 5.7 into communication traffic. Since P_{swap} is fixed, the cost depends only on the number of swaps, and it peaks on Office-Home at $E_{\text{loc}} = 4$, the regime that reorganises the most. Even there, however, the extra cost remains small: 1.04% to 2.80% of the total traffic, or 2.07% to 5.59% of the download stream alone, which is the only one a swap loads.

The protocol may migrate experts in at most 12% of the 50 rounds—the six windows admitted by t_{assign} —and Table 5.7 shows that almost every migration falls in the first of them. Once the local pools match the demand, the admission margin rejects the remaining candidates and the topology stops changing: for most of the run a dynamic client and its static counterpart exchange exactly the same payload. The swap term $N_{\text{swaps}} P_{\text{swap}}$ is thus a one-time cost in our configuration. Figure 5.13 makes both effects visible on the most expensive regime: the upload never moves, and the download rises above the static baseline only where a swap actually lands.

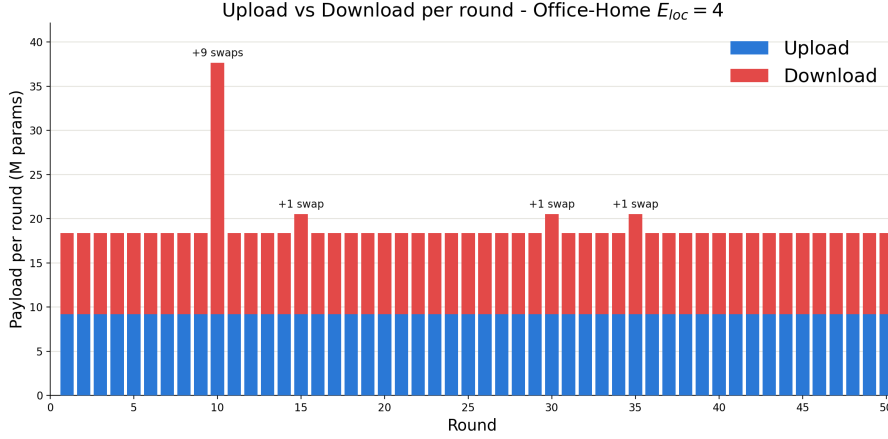


Figure 5.13: **Per-round communication payload under dynamic reassignment**, summed over the eight clients, on Office-Home at $E_{\text{loc}} = 4$ (balanced placement). Each bar stacks the upload onto the download; the annotations report the experts migrating into the federation at that round. The upload is a flat $E_{\text{loc}} \cdot L$ worth of adapters at every round, confirming the one-directional nature of the overhead: only the download reacts to a swap, and it does so by exactly $s_c(t) \cdot P_{\text{swap}}$. Nine of the twelve migrations land at round 10, where the payload momentarily doubles, while the three residual ones barely lift their bar. In the remaining 46 rounds the exchange is byte-for-byte the static one, which is why a transfer as heavy as 8.17 MB per expert still amounts to a few percent of the run.

The final verdict Taken in isolation, moving an expert is expensive: P_{swap} is nearly twice the payload a client exchanges in an ordinary round, and 98% of it is frozen base weights that no PEFT scheme can compress. Read at the level of the run, however, that unit price never compounds. The migrations concentrate in the first swap window, after which the topology stabilises and the remaining rounds are byte-for-byte identical to the static baseline: the transfer is paid once and amortised over the 50 rounds that follow (1.04% to 2.80% of the total traffic). Against a mechanism that reduces the cross-seed variance and makes each configuration reproducible as analysed before, a communication overhead of this order is a negligible price.

5. Experimental Results

Partial-Dynamic Configuration	Swaps (avg over seed)	Extra traffic (MB)	Overhead ↓ (total traffic)	Overhead ↓ (download only)
<i>CIFAR-100</i>				
$E_{\text{loc}} = 3$ Balanced	4.0	32.9	1.04%	2.07%
$E_{\text{loc}} = 3$ Random	5.0	41.1	1.30%	2.58%
$E_{\text{loc}} = 4$ Balanced	7.3	60.1	1.76%	3.51%
$E_{\text{loc}} = 4$ Random	7.0	57.4	1.68%	3.35%
<i>Office-Home</i>				
$E_{\text{loc}} = 3$ Balanced	8.0	65.6	2.07%	4.14%
$E_{\text{loc}} = 3$ Random	9.7	79.5	2.51%	5.01%
$E_{\text{loc}} = 4$ Balanced	11.7	95.5	2.80%	5.59%
$E_{\text{loc}} = 4$ Random	11.3	92.6	2.71%	5.41%

Table 5.10: **Communication overhead of dynamic reassignment**, measured against the matched static partial-replica run (3161.2 MB at $E_{\text{loc}} = 3$ and 3411.2 MB at $E_{\text{loc}} = 4$, upload and download, over 50 rounds and 8 clients). *Extra traffic* is the sum of the two rightmost terms of Equation 5.4 over the eight clients, with the swap counts of Table 5.7; of it, 99.6% is expert migrations and 0.4% telemetry. The last two columns give the same overhead normalised against the total round trip and against the download stream alone, the only one the swaps actually load (↓: the lower the better). Dynamism costs between 1.0% and 2.8% of total communication, scaling with the two drivers that also govern the swap count: a looser local budget and a stronger heterogeneity. Office-Home at $E_{\text{loc}} = 4$, the regime that reorganises most, is the most expensive, and still adds less than 3% of total extra cost.

5.6 Summary of the Findings

This chapter has measured the partial-replica V-MoE along several separate axes: model accuracy, routing behaviour, memory, communication, and stability. Before moving to the conclusions, this section brings the main results together in three summary figures. Each figure shows, at a glance, one of the central findings discussed above.

Convergence is preserved Figure 5.14 compares the server-side Acc@1 of the full-replica reference with the static and dynamic partial replicas at $E_{\text{loc}} = 4$, over the whole fine-tuning. The three curves grow almost together: at every round, the partial runs stay within 1–2 pp of the full replica, and the two partial paradigms reach the same mean accuracy. Reducing the local pool therefore lowers the final accuracy by a small, bounded amount, without changing how the training behaves.

The savings are far larger than the cost Figure 5.15 puts the two sides of the trade-off on the same relative scale: the per-round payload saved by each budget, and the Top-1 accuracy lost in exchange. Once the two quantities share the same unit, the message of RQ1 becomes evident: every configuration saves roughly ten times more bandwidth than the accuracy it gives up, and the extra traffic of the dynamic swaps reduces this saving only slightly.

Dynamism makes the results reproducible Finally, Figure 5.16 breaks the accuracy tables down to the individual placement seeds, connecting each static run with the

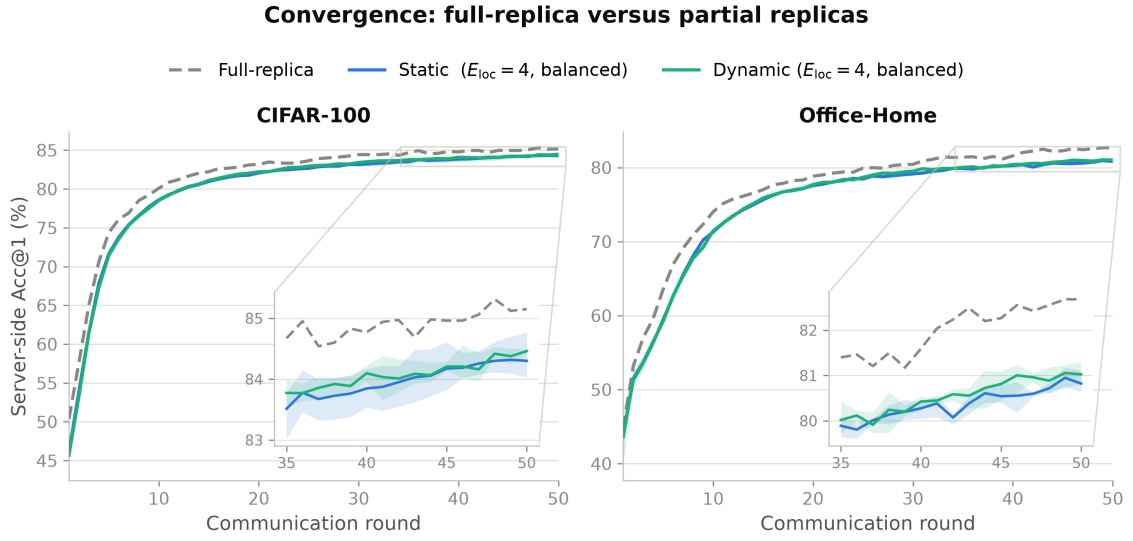


Figure 5.14: **Server-side Acc@1 over the 50 communication rounds** on CIFAR-100 and Office-Home. The plot compares the full-replica reference (dashed) with the static and dynamic partial replicas at $E_{\text{loc}} = 4$ under the balanced placement. Each partial curve is the mean over the 3 placement seeds, and the shaded band shows the minimum and maximum across them; the inset zooms in on the last 15 rounds. The three curves grow almost together: the partial configurations stay within 1–2 pp of the full replica for the whole run, in line with Table 5.6. Static and dynamic reach the same mean accuracy, although on CIFAR-100 the dynamic band is clearly narrower at the end of training—a first sign of the stability result of Figure 5.16.

5. Experimental Results

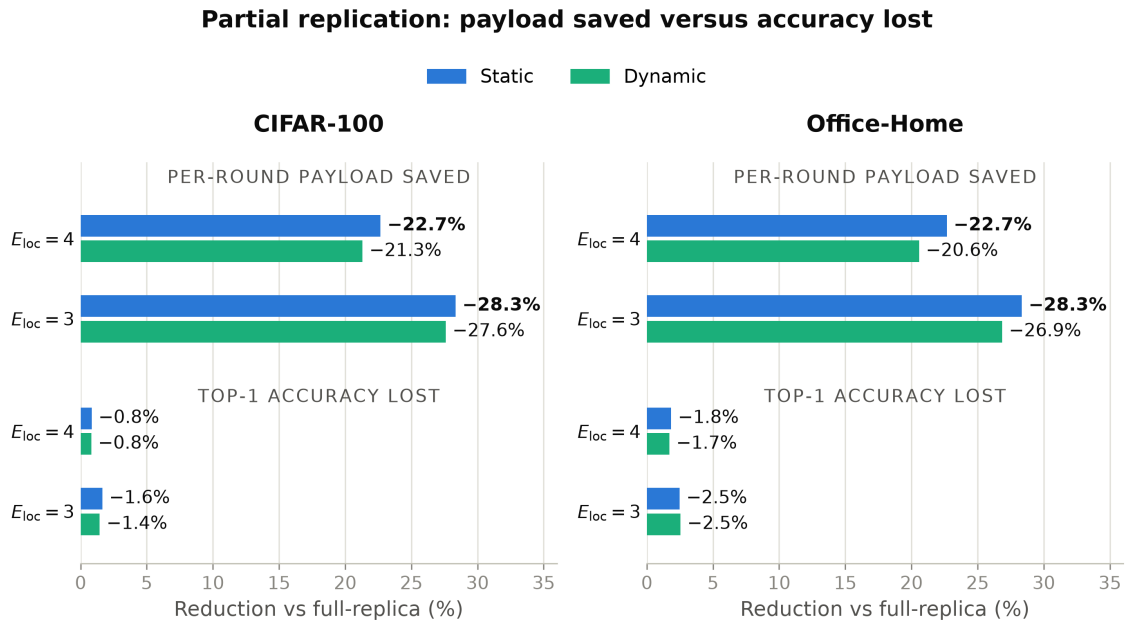


Figure 5.15: **What partial replication saves versus what it costs**, relative to the full-replica reference, keeping the static and the dynamic regimes separate. The upper block shows the per-round communication payload saved at each budget (Table 5.4). In the dynamic bars, the saving already includes the swap overhead of Table 5.10: it costs only 1–3% of the run traffic, which is the small gap between the two bars. The lower block shows the relative Top-1 accuracy lost, with respect to the same reference, by the best configuration of each paradigm and budget (Table 5.6). On the same scale, the imbalance is clear: every budget saves roughly ten times more bandwidth than the accuracy it loses, and the dynamic protocol keeps almost all of this saving.

dynamic run that shares the same seed. This paired view isolates the real contribution of the reassignment protocol: at $E_{\text{loc}} = 4$ the results of the three seeds end up at almost the same value, while at the tight $E_{\text{loc}} = 3$ they remain spread out—the visual counterpart of the budget condition found in the swap analysis.

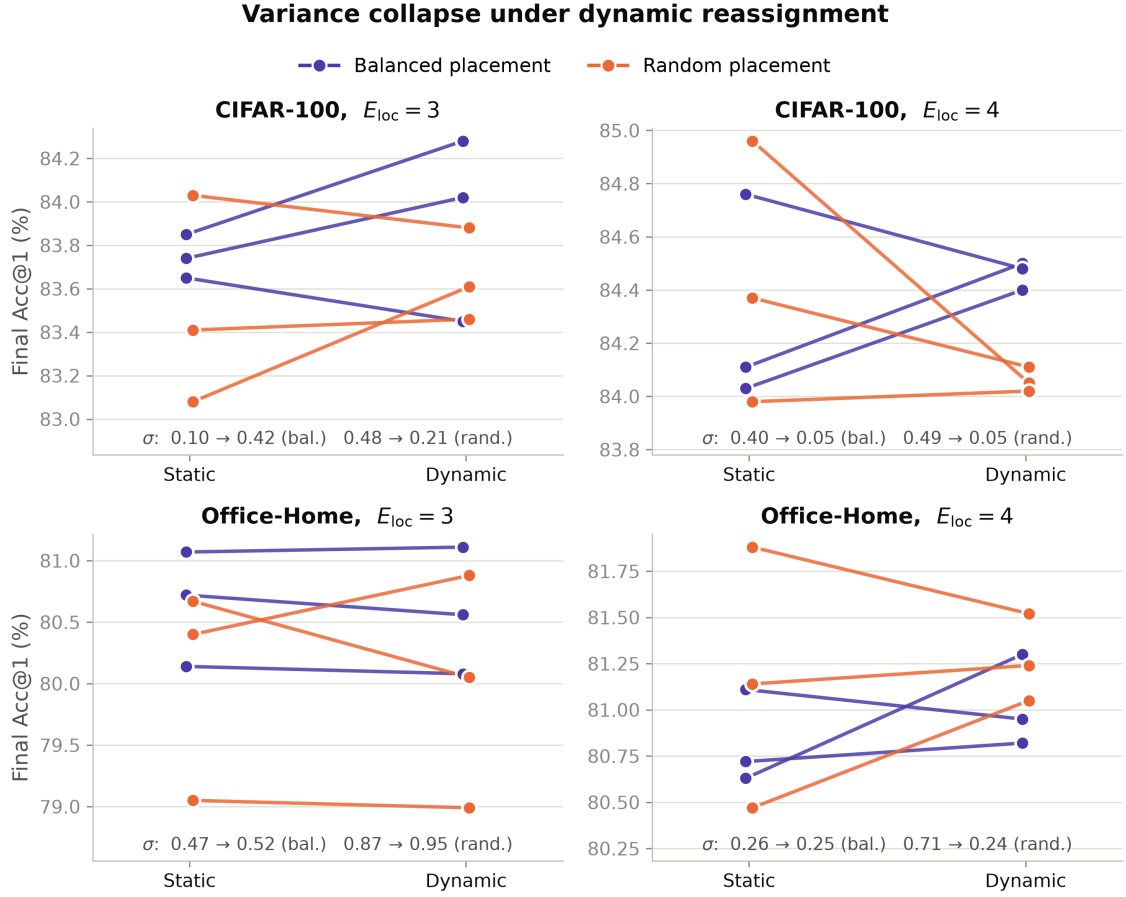


Figure 5.16: **Per-seed final Acc@1, static versus dynamic**, for both budgets and benchmarks. Each point represents a different placement seed, and each segment connects the static and the dynamic run that share the same seed. The slope of each line therefore shows the effect of reassignment alone; the annotations report the standard deviation across seeds on each side. At $E_{\text{loc}} = 4$ the spread collapses (from $\pm 0.40/\pm 0.49$ to ± 0.05 on CIFAR-100) and all runs end at almost the same value, whatever the initial placement. At the tight $E_{\text{loc}} = 3$, instead, the lines stay spread out: the stabilisation needs enough free room for the swaps to complete, as shown in Table 5.7.

Read together, the three figures summarise the chapter: partial replication keeps the convergence of the full model, turns a small accuracy loss into large bandwidth and memory savings, and—once combined with demand-driven reassignment—delivers this trade-off in a reproducible way, regardless of how the experts were assigned at the start.

5.7 Conclusions and Future Work

This thesis investigated whether the partial-replica paradigm—originally designed for LLM pre-training across datacentres—remains viable for the federated fine-tuning of Vi-

5. Experimental Results

sion Mixture-of-Experts (V-MoEs) on resource-constrained edge devices with severely non-IID data. Using the Flower framework, we deployed a LoRA-adapted V-MoE-S/32 with partial-replica distribution and sparsity-aware aggregation scheme. Evaluating this setup under hierarchical label skew (CIFAR-100) and domain skew (Office-Home) yielded clear answers to the three research questions raised in Chapter 1.

RQ1 — Partial replication is cheap in accuracy and valuable in resources

Restricting each client to half—or fewer—of the expert pool costs only 1 to 2 percentage points of Top-1 accuracy with respect to the full replica trained on the same non-IID partitions (Table 5.3). In exchange, the per-round communication payload shrinks by up to 28% and the static on-device memory by up to 36%, while the skip-token mechanism additionally lowers the compute actually executed on the device (Tables 5.4 and 5.5). The loss is contained by three reinforcing factors: the sparsity-aware aggregation, which shields rarely hosted experts from dilution; the redundancy of top-2 routing and the residual pathway, which give most tokens a local fallback; and the bounded sparse footprint of the V-MoE-S/32, whose experts occupy a limited fraction of the network. The same architectural property that caps the penalty also caps the absolute savings—but the *Every-2* projection shows both scale directly with MoE density, reaching a 41% payload reduction under a denser placement: the more an architecture leans on its experts, the more partial replication has to offer.

RQ2 — Ghost demand is constant, and a static assignment cannot absorb it

The ghost-activation analysis returned the sharpest behavioural finding of this work. Ghost rates sit at the structural share of missing experts ($\sim 62\%$ at $E_{\text{loc}} = 3$, $\sim 50\%$ at $E_{\text{loc}} = 4$) and stay flat across all 50 rounds of fine-tuning. The demand is not transient: the pre-trained router, bound by its prior and by the minimal learning rate that protects it from catastrophic forgetting, keeps requesting the same experts regardless of their local availability. Under a static assignment the mismatch is therefore permanent. Whichever experts a client is denied at round 0, it keeps demanding until round 50; no amount of training dissolves it. This constancy cuts both ways. It makes ghost demand measurable and predictable, which is what keeps the partial replica viable. But it also leaves a single lever to act on: the placement itself. If the demand will not move towards the topology, the topology must move towards the demand—hence the dynamic expert reassignment addressed by the final question.

RQ3 — Dynamic reassignment stabilises; it does not elevate

The server-orchestrated dynamic reassignment protocol aligns the topology with demand—ghost rates fall by up to ~ 7 pp on Office-Home, and the final replication of each expert correlates strongly with its routing demand ($r = 0.95$)—but this alignment does not raise the accuracy ceiling: the mean Top-1 moves by fractions of a point, because the loss of partial replication stems from *absent* capacity, which a capacity-preserving swap cannot create. What the mechanism buys is *predictability*. At $E_{\text{loc}} = 4$ the cross-seed standard deviation collapses (from $\pm 0.40/\pm 0.49$ to ± 0.05 on CIFAR-100), neutralising the dependence on the initial placement—provided the local budget leaves enough room for the swaps to complete. And the price is marginal: the migrations concentrate in the first reassignment window and amortise to 1–3% of the total communication. Dynamic reassignment is thus best understood as cheap insurance against an unlucky initial assignment, not as an accuracy mechanism.

Taken together, the findings compose into a practical recipe: partial expert replication bounds the footprint, parameter-efficient adaptation bounds the traffic, sparsity-aware aggregation preserves the updates, and demand-driven reassignment makes the result reproducible. On the tested benchmarks, this recipe brings a sparse vision model within the memory, compute, and bandwidth constraints of a federation of edge devices at a well-characterised, single-digit accuracy cost. More broadly, it decouples the local hardware limit from the global model size: the federation can grow its shared model beyond what any single participant could host.

Future work Each of the findings above points to a natural direction forward:

- **Regimes where dynamism can return accuracy.** On our stationary partitions, reassignment returned stability but not accuracy. The regimes where it could return both are those in which the initial placement is genuinely wrong or becomes wrong over time: stronger heterogeneity, clients dropping out mid-training, and *non-stationary* local distributions that drift during training (where no fixed assignment can remain correct for the whole run). The natural extreme is *continual learning*, where clients train on an open-ended stream and the semantics of their local data can shift radically: new classes appear, and whole domains replace old ones. Every such shift redraws the expert demand, turning periodic reassignment from a one-off correction into the mechanism that keeps the topology tracking the federation’s demand.
- **Denser expert placements.** The *Every-2* projection indicates that both the savings and the risks of partial replication grow with MoE density. Validating the paradigm on architectures that host experts in every other block—where the sparse component dominates the network—would test whether the protective factors identified under RQ1 continue to hold. The caveat is that the two sides scale together: with experts in more blocks, tokens meet ghost experts more often, and more of the computation is skipped. A study of denser architectures must therefore measure the whole trade-off—memory, activations and bandwidth saved on one side, accuracy lost on the other. The number of experts per layer is a second axis: our pool of $E_{\text{tot}} = 8$ is small, and most MoE architectures host many more, which multiplies the possible E_{loc} configurations to explore.
- **Heterogeneous local budgets.** Our federation is deliberately uniform, with every client granted the same E_{loc} . Real cross-device fleets are heterogeneous: allowing per-client budgets matched to each device’s memory, with the reassignment protocol arbitrating experts across tiers, would turn the constant-footprint constraint into a resource-allocation policy.
- **Cheaper migrations.** The cost analysis showed that 98% of an expert transfer is frozen base weights. Quantising or compressing the migrating expert—acceptable precisely because those weights are frozen and receive no gradient—would shrink the only non-negligible cost that dynamism carries. In our stationary runs, that cost is already amortised to 1–3% of the total traffic; compression would pay off in the regimes above, where swaps are no longer confined to the first reassignment window but recur throughout the run.

Bibliography

- [1] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022.
- [2] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 20–22 Apr 2017.
- [3] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.
- [4] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarczyk, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.
- [5] Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Scaling vision with sparse mixture of experts, 2021.
- [6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.
- [7] Lorenzo Sani, Zeyu Cao, Meghdad Kurmanji, Alex Iacob, Andrej Jovanovic, Yan Gao, Wanru Zhao, and Nicholas D. Lane. Fomoe: Breaking the full-replica barrier with a federation of moes, 2026.
- [8] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.
- [9] Zihan Fang, Zheng Lin, Senkang Hu, Yanan Ma, Yihang Tao, Yiqin Deng, Xianhao Chen, and Yuguang Fang. Hfedmoe: Resource-aware heterogeneous federated learning with mixture-of-experts, 2026.
- [10] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchu Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque

- de Gusmão, and Nicholas D. Lane. Flower: A friendly federated learning research framework, 2022.
- [11] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feed-forward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.
 - [12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.
 - [13] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.
 - [14] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.
 - [15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
 - [16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.
 - [17] Lorenzo Sani, Alex Iacob, Zeyu Cao, Royson Lee, Bill Marino, Yan Gao, Dongqi Cai, Zexi Li, Wanru Zhao, Xinchu Qiu, and Nicholas D. Lane. Photon: Federated llm pre-training, 2026.
 - [18] Arthur Douillard, Qixuan Feng, Andrei A. Rusu, Rachita Chhaparia, Yani Donchev, Adhiguna Kuncoro, Marc’Aurelio Ranzato, Arthur Szlam, and Jiajun Shen. Diloco: Distributed low-communication training of language models, 2024.
 - [19] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrill, Quentin De Bouthillier, Andrea Gesmundo, Jean-Philippe Boytsov, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.
 - [20] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, 2021.
 - [21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.
 - [22] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.
 - [23] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

BIBLIOGRAPHY

- [24] Hemanth Venkateswara, Jose Eusebio, Shayok Chakraborty, and Sethuraman Panchanathan. Deep hashing network for unsupervised domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5018–5027, 2017.
- [25] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.
- [26] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.