Privacy-Aware Distributed Machine Learning

by

Evan W.R. Madill

A Thesis submitted to the Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science
The University of Manitoba
Winnipeg, Manitoba, Canada

July 24, 2025

Thesis advisor | Co-advisor | Author
Dr. Carson K. Leung | **Dr. Sara Rouhani** | **Evan W.R. Madill**

Privacy-Aware Distributed Machine Learning

Abstract

Split Learning (SL) is a promising framework for distributed machine learning in resource-constrained and privacy-sensitive settings. However, vanilla SL deployments have limitations in communication overhead, privacy leakage from intermediate activations, accommodating heterogeneous client capabilities, and integrating with federated strategies. This thesis addresses these challenges through four primary contributions. First, we address communication efficiency by proposing a Vector Quantized Variational Autoencoder (VQ-VAE) framework using Lookup-Free Quantization (LFQ) for compressing intermediate features. This eliminates the need for codebook transmission, with favorable rate-distortion trade-offs. Second, we introduce PRISM (Privacy Router with Integrated Spatio-Channel Masking), an information router and optimization strategy for enhancing privacy in SL. PRISM adopts a U-shaped network, fine-grained masking, a local bypass stream, and disentangled optimization to reduce information leakage to the server while preserving task utility. Third, we propose Split Learning with Frozen Clients (SL-FC), a strategy designed for heterogeneous environments. SL-FC studies how ultra-resource-constrained "frozen" clients, capable only of forward propagation, can contribute their data to server-side training, demonstrably improving overall model performance. Finally, we evaluate the Muon optimizer for use as an outer-loop, or federated optimizer. Muon orthogonalizes momentum updates, aiming to improve convergence speed and stability compared to standard federated optimizers like FedAvgM and FedAdam. This thesis provides theoretical analyses, rate-distortion bounds, convergence guarantees, and empirical evaluations to demonstrate the effectiveness of the proposed methods. Tasks involve image classification, segmentation and model training under non-IID constraints. Collectively, these contributions improve the practicality, and privacy over existing split and federated learning systems.

Acknowledgements

First and foremost, I wish to express my sincere gratitude to my supervisor (Dr. Carson K. Leung) and my co-supervisor (Dr. Sara Rouhani, who recently moved to École de technologie supérieure (ÉTS) Montréal). Dr. Leung's encouragement was pivotal in my decision to pursue a Master's degree. I thank Dr. Rouhani for introducing me to the fascinating field of federated learning, which sparked my initial interest and set me on this research path. Together, their guidance, expertise, and seemingly infinite patience were invaluable throughout the process of completing this thesis. They also generously provided opportunities to engage with industrial projects which broadened my experience.

I would also like to extend my gratitude to my examining committee members (Dr. Noman Mohammed and Dr. Shaowei Wang) for their insightful feedback and rigorous analysis during the review of my thesis. Thanks Dr. Christopher J. Henry for chairing my defence. My appreciation also goes to the University of Manitoba and my supervisor's NSERC & Mitacs grants for providing the resources and support that made this research possible.

Lastly, I am deeply thankful to my parents. Their unwavering support, encouragement, and belief in me, especially during times progress was slow, and the journey took longer than anticipated.

Evan W.R. Madill

B.Sc.(Hons.), Comp. Sci./Physics, The University of Manitoba, 2021

The University of Manitoba
July 24, 2025

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence.
ARL	Adversarial Representation Learning.
BPP	Bits Per Pixel.
CNN	Convolutional Neural Network.
DNN	Deep Neural Network.
DP	Differential Privacy.
FL	Federated Learning.
FSL	Federated Split Learning.
GPU	Graphics Processing Unit.
IID	Independent and Identically Distributed.
IoT	Internet of Things.
KD	Knowledge Distillation.
LFQ	Lookup-Free Quantization.
LLM	Large Language Model.

LPIPS	Learned Perceptual Image Patch Similarity.
MI	Mutual Information.
mIoU	mean Intersection over Union.
ML	Machine Learning.
PPML	Privacy-Preserving Machine Learning.
PQ	Product Quantization.
PRISM	Privacy Router with Integrated Spatio-channel Masking.
PSL	Parallel Split Learning.
PSNR	Peak Signal-to-Noise Ratio.
R-D	Rate-Distortion.
SGD	Stochastic Gradient Descent.
SGDM	Stochastic Gradient Descent with Momentum.
SL	Split Learning.
SL-FC	Split Learning with Frozen Clients.
SSIM	Structural Similarity Index Measure.
STE	Straight-Through Estimator.
SVD	Singular Value Decomposition.
VAE	Variational Autoencoder.
VQ	Vector Quantization.
VQ-VAE	Vector Quantized Variational Autoencoder.
ZKP	Zero Knowledge Proof.

Chapter 1

Introduction

Distributed machine learning has become essential for training models on decentralized data. Driven by the increasing volume of data generated at the network edge, the computational capabilities of edge devices, concerns regarding data privacy, and communication costs-trends have been amplified by the rise of large-scale models. Collection of edge data for centralized training is often impractical or undesirable due to privacy regulations, communication bottlenecks, and latency requirements, a challenge addressed in domains ranging from healthcare and mobile services to agriculture [YAE+18, XGS+21, DMM+22]. Federated Learning (FL) [MMR+17, KMA+21] and Split Learning (SL) [GR18, VGSR18], along with hybrid approaches such as SplitFed [TZEM21, TACS22], offer compelling alternatives enabling collaborative model training across distributed data sources without sharing raw data. FL involves local model training and server-side update aggregation, allowing for client parallelism, while SL partitions models between a client and server, with clients sharing intermediate activations to be completed server-side.

Despite their advantages, the practical deployment of these distributed paradigms, particularly for training models in complex, real-world heterogeneous environments, faces hurdles limiting their effectiveness and adoption. This thesis identifies and addresses four primary, often interconnected challenges:

- **Communication Overhead:** The transmission of intermediate activations in SL, especially for deep models or high-resolution data, can impose prohibitive

communication burdens, requiring efficient compression methods to preserve task-relevant information across client-server boundaries [ADK+23, OLBJ25].

- **Privacy Leakage:** While raw data remains localized, shared intermediate activations in SL [GBDM20, EKÇ22] and model updates in FL [ZH20] can inadvertently reveal sensitive information about client data, necessitating active privacy-enhancing technologies that go beyond basic data partitioning.

- **Resource Heterogeneity:** Distributed systems typically consist of clients with diverse computational, memory, and energy capacities [ZLL+18]. Protocols rigidly requiring full participation exclude less capable devices, leading to underutilization of data and potential model biases. Accommodating heterogeneity is crucial for inclusivity and robustness.

- **Optimization Challenges:** Training models across distributed, potentially non-IID datasets is inherently difficult [RCZ+21]. Standard optimization algorithms like FedAvg can struggle with client drift and convergence stability, while directly applying centralized adaptive methods (e.g., Adam) may not be optimal or efficient in the federated context. This is compounded by data privacy driving the need to share less information, requiring more sample-efficient methods.

These challenges require more than isolated fixes. Solutions must address the interplay between communication, privacy, resource constraints, and optimization, particularly within the design space offered by Split Learning and its Federated variants.

1.1 Foundations of Privacy-Aware Machine Learning

Privacy-preserving machine learning (PPML) aims to protect sensitive data while allowing multiple parties to jointly train models. Two notions of privacy in this context are **input privacy** and **output privacy** [TBC+24]. Input privacy means a

system can compute on someone's data without revealing that data to others, allowing an intermediary to deliver its contents without explicitly reading them. Output privacy means the observed result of a computation (e.g., a model update or prediction) should not reveal information about the original inputs. In other words, you can contribute data to a collaborative model without fear that the model's outputs can be reverse-engineered to expose your private data. Ensuring both input and output privacy is critical in federated learning, split learning, and related techniques. However, achieving strong privacy often introduces a design tension, the more we hide or encrypt the data flow, the harder it becomes to verify and validate what is being computed. This tradeoff between privacy and verifiability lies at the heart of PPML system design.

This tradeoff is demonstrated clearly in FL [MMR+17]. FL keeps training data on user devices, sending only model updates to an aggregator. Techniques like **secure aggregation** [BIK+17] provide input privacy by cryptographically masking each client's update so the server sees only their aggregate sum. While this protects individual data contributions, it also obscures the learning process, such that the server cannot inspect or validate any single update semantically. This makes it difficult to apply robust validation or anomaly detection on contributions, such as *Krum* and *Multi-Krum* [BEGS17] which operate by selecting a subset of updates that are closest to each other and down-weighting outlier updates in order to mitigate malicious outliers. Such methods require access to each client's model update. If all updates are encrypted or hidden from the server, it becomes difficult to verify which updates are trustworthy. Zero knowledge proofs (ZKPs) are a promising direction [MLZL24, WDS+24], but remain limited to syntactic properties (e.g., value ranges, adherence to computation rules). Simply providing input/output privacy alone may conflict with the need for participants to verify the accuracy and trustworthiness of the results [TBC+24]. This emphasizes the challenge of maximizing privacy of data flows limits our ability to validate a model's integrity, whereas revealing these data flows risks exposing sensitive information.

Similar challenges arise with output privacy. Even when raw data remains local on-device, the model's outputs or updates can inadvertently leak information about

the training data if not protected. Attacks like inference or reconstruction can use model gradients or predictions to learn about individual examples [GBDM20, EKÇ22]. To mitigate this, mechanisms such as **differential privacy (DP)** [Dwo06], which inject statistical noise to the model updates, are employed. DP provides a quantified privacy guarantee on outputs, which ensures that any single data point has negligible influence on the model's outputs. This noise can hide the presence or effect of a client's data in the aggregated results, which reduces the risk of leakage. This however, comes at the expense of model utility and complexity. The privacy-utility tradeoff implies stronger output privacy may degrade accuracy or obscure behavior, while less restriction means easier evaluation and debugging but risks privacy. When proposing production systems, a carefully chosen set of constraints is needed, informing on how one balances privacy while allowing the system to be audited or validated for correctness.

Structured transparency is one framework which has emerged to navigate this design space in a principled way [TBC+24], and aligns with a broader push towards explainable and trustworthy artificial intelligence (AI), where the goals are to quantify interpretability, understand system behavior, and ensure responsible deployment [SB19, ADDS+20]. Privacy and transparency are not entirely mutually exclusive, which is emphasized in building systems with well-defined information flows, with some work even exploring interpretable federated learning directly [RHCJ21]. The core idea is that *privacy safeguards* should be combined with *verification mechanisms* (inputs are authentic and outputs are correct) under an appropriate *governance structure*. This perspective identifies five criteria for any collaborative learning flow: input privacy, output privacy, input verification, output verification and flow governance. Input/output privacy focus on what needs to be hidden, while input/output verification focus on what needs to be revealed to assure the results are reliable. Flow governance provides oversight, specifying *who* is allowed to know or do *what* at each stage of the process. In practical terms, this means designing systems where the training procedure has an audit trail or evidence of correctness. A securely computed model may come with cryptographic proofs or statistical certificates showing it was trained on legitimate inputs without leakage.

Split Learning (SL) [GR18] is one such tool serving as an important design point in the privacy, verification tradeoff space. Like FL, split learning allows multiple parties to collaboratively train a model without sharing raw data. In SL, the model's architecture is split between two parties, a client and server. Each client trains the first layers of the neural network on local data up to some 'cut layer' sending only the intermediate activations to the server, which continues the forward and backward pass. The server never sees the original inputs enforcing input privacy architecturally, and further providing a form of model privacy since server weights are not shared. Since the server is directly involved in the training pipeline, this creates the opportunity for built-in validation and oversight. The main benefit of SL's design is its ability to accommodate resource heterogeneity. Since a large portion of the model's computation is handled by the server, the computational burden on each client is greatly reduced, allowing even devices with limited processing power or bandwidth to participate, requiring only a few layers to be trained with relatively small activation tensors. The primary challenge of split learning lies in the bundling problem [TBC+24], where task related information requires the context of sensitive information. This is often addressed through aggressive activation pruning, which attempts to disentangle task and sensitive information. While this does not make strong privacy guarantees (e.g., with differential privacy), it has shown to be an empirically promising form of obfuscation. In summary, by splitting the model, SL importantly manages to uphold input privacy, yet it leaves more room to incorporate verification techniques and efficient coordination than some fully decentralized approaches.

The spectrum from federated learning to split learning illustrates some of the tradeoffs between privacy and verifiability in distributed machine learning (ML). Federated learning leans toward the privacy end (data never leaving the device), whereas split learning allows a more diverse set clients to collaborate, especially under extreme resource constraints. This design space is a crucial part of privacy-preserving systems, and should not operate as black boxes. In the chapters that follow, we build on this idea primarily from the perspective of a resource constrained client, addressing some of the common challenges present in distributed learning systems.

1.2 Federated Learning and Split Learning

Motivated by privacy regulation, bandwidth, and latency constraints, distributed learning aims to train shared models on data that remains on edge devices. Two complementary paradigms have emerged: Federated Learning (FL) and Split Learning (SL). Both avoid sharing raw inputs, but they expose different interfaces to the server and make different system trade-offs in computation, communication, and privacy [MMR+17, GR18, VGSR18, TZEM21, TACS22].

Federated Learning (FL). FL considers K clients with local datasets $\{D_k\}_{k=1}^{K}$, sizes $\{n_k\}$, and a global objective

$$F(w) \;=\; \sum_{k=1}^{K} \frac{n_k}{N} F_k(w), \qquad F_k(w) \;=\; \mathbb{E}_{\xi\sim D_k}\big[\ell(w;\xi)\big], \quad N = \sum_k n_k, \tag{1.1}$$

where w are model parameters. In a typical round t, a subset $\mathcal{S}_t$ of clients performs S local SGD steps starting from the broadcast global model w_t, yielding local models $\{w_{t+1}^k\}_{k\in\mathcal{S}_t}$. The server aggregates

$$w_{t+1} \;=\; \sum_{k\in\mathcal{S}_t} \frac{n_k}{\sum_{j\in\mathcal{S}_t} n_j}\, w_{t+1}^k, \tag{1.2}$$

as in Federated Averaging [MMR+17]. Communication is periodic and coarse-grained: each active client transmits and receives $\Theta(|W|)$ parameters per round (potentially with compression). This allows for strong *input privacy*, applicable at mobile/IoT scale, and private inference allows for complete privacy at test time. Several challenges include:

1. communication of high-dimensional updates over many rounds.
2. system heterogeneity (stragglers, dropouts, resource limits).
3. statistical heterogeneity (non-IID D_k leading to client drift).
4. privacy surface on model updates (susceptible to inference or reconstruction unless protected, e.g., via secure aggregation or differential privacy) [BIK+17, Dwo06].

5. verification or attestation of model updates (e.g. [BEGS17]), preventing malicious clients from poisoning or hijacking, without requiring raw gradients or parameters.

Split Learning (SL). SL partitions a network into a client-side submodel $M_C(\cdot;\theta_C)$ and a server-side submodel $M_S(\cdot;\theta_S)$ at a cut layer. For a labeled sample (x, y), the client computes the smashed data

$$z \;=\; M_C(x;\theta_C), \tag{1.3}$$

sends z to the server, which completes the forward pass $\hat{y} = M_S(z;\theta_S)$, evaluates a loss $L(\hat{y}, y)$, and backpropagates to obtain $\nabla_z L$. The server returns $\nabla_z L$ to the client, which finishes backprop through M_C to update θ_C. The server updates θ_S with its own gradients [GR18, VGSR18]. Raw inputs (and often labels) remain on-device, providing architectural input privacy, and clients only need to store/compute a prefix of the model, lowering memory/compute barriers for resource-constrained devices.

SL's primary trade-offs differ from FL in how communication is handled: for batch size B and cut-layer activation dimension $q = \dim(z)$, a vanilla SL iteration exchanges $\Theta(B\,q)$ values each step (activations forward; gradients backward). Communication, while fine-grained, is more frequent, and scales with dataset sizes n_k. This can exceed FL's per-round cost unless q is small, or activations are compressed (cf. Chapter 2). Another practical consideration is parallelism: a naive SL pipeline processes client batches through the same server-side tail, which can serialize training. Systems variants (micro-batching, pipelining, or multi-instance server tails) increase throughput at the expense of server resources. Finally, the privacy surface shifts: intermediate features z may bundle task-relevant and sensitive factors, allowing inversion or attribute inference; activation pruning, masking, and noise can empirically reduce leakage (cf. Chapter 3) but often without formal DP guarantees.

Comparisons and Limitations. Both paradigms keep raw data local, but expose different interfaces:

- **Client workload:** FL requires fully local training, while SL requires only partial (up to the cut) model training, enabling weaker devices to participate.
- **Communication:** FL exchanges model updates each round ($\Theta(|W|)$). SL uses activation and gradient exchanges each step ($\Theta(B\,q)$). FL allows a variable number of local steps S trading accuracy vs. fewer rounds. SL varies model cut depth, trading client compute vs. activation size q.
- **Privacy surface:** FL exposes model deltas/gradients (protectable via secure aggregation and DP [BIK+17, Dwo06]). SL exposes intermediate representations z (which can be mitigated via DP, pruning/masking and encryption).
- **Transparency & verification:** FL's server typically observes only aggregated updates, limiting semantic validation. SL's server participates in the forward/backward pass and can monitor activations/outputs, allowing verification at the cost of a broader attack surface on z.

Hybrid paradigms. FL and SL are not mutually exclusive. SplitFed [TACS22] combines SL's partitioning with FL-style aggregation across client models. Client models are trained in parallel and periodically averaged, while a shared server-side suffix provides acceleration and central compute [TZEM21]. Such hybrids can alleviate vanilla SL's sequential bottleneck while preserving a lower client workload, and can be complemented with privacy mechanisms (e.g., secure aggregation on client models, or DP on server-side updates). Related designs (e.g., federated split learning, parallel split learning [ZPT+23, CW22]) explore synchronization frequency, label placement ("U-shaped" SL [VGSR18]), and activation handling to balance utility, privacy, and throughput.

Relation to this thesis. This thesis builds on these foundations. Chapter 2 addresses SL's communication with learned discrete bottlenecks; Chapter 3 studies privacy leakage from shared activations and proposes masking/routing mechanisms; Chapter 4 targets resource heterogeneity (e.g., frozen clients that only forward-

propagate); and Chapter 5 evaluates the Muon optimizer for stability and speed in federated settings.

1.3 Thesis Statement

This thesis argues that the practical limitations of Split and Federated Learning systems concerning communication overhead, privacy vulnerability, resource heterogeneity, and optimization effectiveness can be largely mitigated through a combination of novel techniques spanning feature compression, privacy-preserving architectural design, adaptive client participation strategies, and structurally-aware optimization algorithms. This work aims to improve the feasibility, performance, and security of deploying modern machine learning models in distributed, resource-constrained, and privacy-sensitive environments. Specifically, this research aims to answer the following questions:

1. *Can we quantize activations without transmitting a codebook?*: Apply a learned, discrete quantization schemes (VQ-VAE + LFQ) that minimize communication overhead without requiring codebook transmission, and maintaining high task utility.

2. *Can we mask only the sensitive bits while preserving task utility?*: Explore how information can be routed through architectural innovations and disentangled optimization strategies that provide empirical privacy against server-side inference attacks, without sacrificing performance.

3. *Can ultra-resource-constrained sensors still contribute signal?*: Adapt Split Learning to include clients that cannot participate fully, improving model performance across heterogeneous environments.

4. *Can we apply the Muon optimizer to a federated/outer-loop setting?* Investigate matrix-aware optimization approaches for improved training dynamics and more stable/efficient convergence than standard coordinate-wise methods.

By answering these questions, this thesis presents four distinct contributions that collectively advance distributed machine learning in privacy-aware settings, facilitating more practical and effective deployments.

1.4 Thesis Organization

The remainder of this thesis is organized as follows:

- **Chapter 2: Compression Mechanisms in Split Neural Networks** addresses the challenge of communication overhead. It proposes and analyzes a Vector Quantized Variational Autoencoder (VQ-VAE) framework employing Lookup-Free Quantization (LFQ) for compressing intermediate features in SL. Theoretical analysis based on rate-distortion theory and empirical evaluations demonstrate its effectiveness.

- **Chapter 3: Privacy Mechanisms in Split Learning** tackles the problem of information leakage from shared activations. It introduces PRISM (Privacy Router with Integrated Spatio-Channel Masking), a novel architecture and disentangled optimization strategy using fine-grained masking and a local bypass stream to enhance privacy and preserve utility.

- **Chapter 4: Resource Constraints in Split Learning** explores solutions for handling client resource heterogeneity. It proposes Split Learning with Frozen Clients (SL-FC), a framework allowing ultra-resource-constrained devices to contribute data via forward-pass only participation, demonstrating significant performance benefits over client exclusion.

- **Chapter 5: Federated Optimization** investigates improvements to the server-side optimization process. It evaluates the Muon optimizer, which applies layer-wise orthogonalization to momentum updates, analyzing its convergence properties and comparing its characteristics to standard federated optimizers.

- **Chapter 6: Conclusions** We first synthesize the findings from the four contributions, discusses their broader implications and interconnections (particularly

within a unified SplitFed framework), acknowledges the overall limitations of the work, and outlines promising directions for future research that integrate these complementary techniques. Then, we give a concise overview of the thesis contributions and conclusions.

Together, these chapters present a systematic approach to overcoming common obstacles in distributed learning, aiming to make technologies like Split Learning more efficient, private, and performant.

Chapter 2

Compression Mechanisms in Split Neural Networks

2.1 Introduction

In split neural networks (also known as *split learning* or *collaborative inference* [GR18]), an input is processed by an edge-side sub-network up to a bottleneck layer, and the intermediate features are transmitted to a server-side sub-network for final inference. Compressing these intermediate features is crucial for communication efficiency, especially as model sizes continue to grow into the billions of parameters, making the transmission of uncompressed activations entirely infeasible [DDM+23]. A popular use case for split inference is computer vision in internet of things (IoT), and smart devices transmitting information feeds to the cloud. Traditional image/video codecs (JPEG, H.264, etc.) can be suboptimal for this task, as they optimize human perceptual quality rather than preserving task-relevant information [ADK+23]. Recent approaches have proposed a learned feature compression jointly optimizing for task performance and compression rate. Notably, Ahuja *et al.* [ADK+23] cast the feature compression problem in split DNNs as a variational autoencoder (VAE) optimized for rate-distortion trade-offs, yielding improved accuracy at lower bitrates. Another line of work (e.g., Oh *et al.*'s work [OLBJ25]) introduces an adaptive feature-wise compression in split learning, dropping less-informative feature channels and

quantizing the rest with data-dependent precision. These methods reduce bandwidth while maintaining accuracy.

While feature compression is integral to reducing communication overhead, other strategies such as local contrastive loss [CSS+21] and distillation [CW22] also target overhead reduction in distributed learning. Importantly, combining payload compression with these methods is not mutually exclusive and can yield further cost savings. A critical aspect in these systems is the inherent trade-off between communication efficiency and computational load. For instance, in SL, the communication payload scales with the activation tensor size ($B \times q$), meaning deeper cuts can shrink communication costs but consequently push more (convolutional) compute to the client device. Both vanilla FL and SL offer simple knobs to manage this balance, such as the number of local client steps k in FL, which affects client drift, or the choice of cut layer in SL, which dictates client-side computation. The optimal configuration often depends on the specific scenario, whether dealing with edge devices that have idle graphics processing unit (GPU) cycles but limited uplink, or ultra-constrained IoT sensors where even early convolutions may need to be offloaded. Table 2.1 provides a comparative summary of these communication and computation costs for popular methods, and Figure 2.1 visually illustrates some of these cost dynamics related to federated and split learning.

In this work, we present a Vector Quantized VAE (VQ-VAE) framework with Lookup-Free Quantization (LFQ) [YLG+24] for compressing splitNN features. Our approach combines discrete latent representation learning with a communication-efficient coding scheme eliminating the need to transmit any codebook. Instead, the quantization is *lookup-free*: each latent dimension is directly quantized to a small set of values, and only the indices are sent [YLG+24]. This allows the decoder to interpret the compressed latent without receiving a separate codebook, unlike standard vector quantization. We formalize the **encoder** $\rightarrow$ **quantizer** $\rightarrow$ **decoder**, and frame the compression problem in information-theoretic terms, with rate-distortion theory and mutual information. In particular, we analyze how the VQ-VAE with LFQ achieves an efficient rate-distortion trade-off, and we establish bounds on rate efficiency relative to theoretical optima. We also discuss convergence properties of the learning algo-

Table 2.1: Amortized time cost analysis. N: devices, B: batch size, k: local steps per round, q: cut-layer activation size, $|W|$: model parameters, β: client-side model fraction, ρ compression fraction, t: client step time, T: server step time, U: aggregation time, D: distillation loss time, Q: quantization time. We assume a standard round-robin for vanilla SL.

Algorithm	**Comm. cost**	**Compute cost (client)**	**Compute cost (server)**
FL [MMR+17]	$2\|W\|$	kt	U
SL [VGSR18]	$2kBq + \beta\|W\|$	$k\beta t$	$Nk(1-\beta)T$
SplitFed [TACS22]	$2kBq + 2\beta\|W\|$	$k\beta t$	$Nk(1-\beta)T + \beta U$
AdaSplit [CSS+21]	$2kBq$	$k\beta t$	$N(1-\beta)T$
PSL [CW22]	$2kBq$	$k\beta t$	$Nk(1-\beta)T + D$
FSL [ZPT+23]	$2kBq + 2(1-\beta)\|W\|$	$k\beta t$	$Nk(1-\beta)T + (1-\beta)U$
SL + Compression	$(1+\rho)kBq + \beta\|W\|$	$k[\beta t + Q_c]$	$Nk[(1-\beta)T + Q_s]$

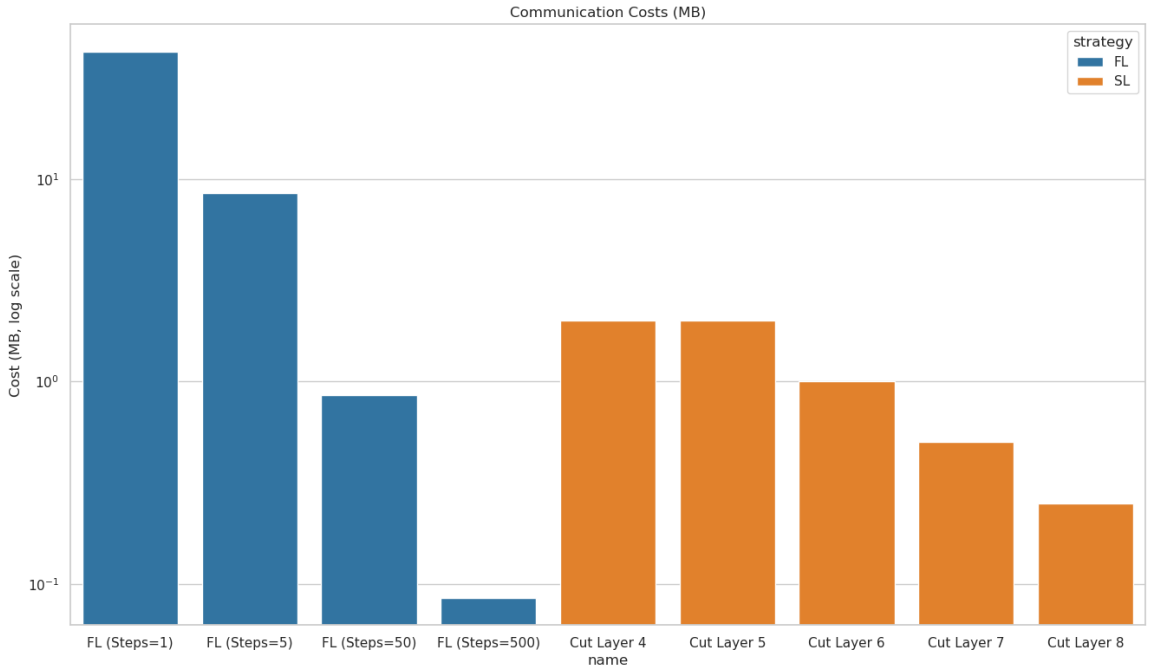

Figure 2.1: Amortized per-step communication overhead to train a ResNet18, with an image size of 128. Steps indicates the number of mini-batches processed each aggregation. Higher steps is more communication efficient, although introduces client drift.

rithm (which alternates between continuous encoder/decoder updates and discrete code assignments). Further, we compare our method to prior feature compression techniques, including the Neural Rate Estimator approach for learned bottlenecks [ADK+23] and the Adaptive Feature-Wise Compression strategy [OLBJ25], demonstrating improvements in communication efficiency. Unlike prior learned quantizers that might require transmitting codebooks or side-information, our method requires only infrequent transmission of decoder parameters (during training), not per-batch codebooks. Additionally, since quantization occurs per-sample, it remains orthogonal to cut layer or batch-level scheduling, and may be layered on dynamic strategies like AdaSplit [CSS+21] and FedLite [WQR+22]. Finally, we examine optimization aspects such as training stability and the role of the vector quantization (VQ) commitment loss, connecting to prior analyses of VQ-VAEs. The following sections provide a detailed construction of our approach.

2.2 Background

In a split deep neural network (DNN), the bottleneck layer at the partition point is trained to produce a lower-dimensional feature z that can be efficiently compressed [ADK+23]. The goal is to minimize the communication rate (in bits per sample) while preserving task-relevant information (minimizing distortion or accuracy loss). This is naturally an information-theoretic rate-distortion (R-D) optimization problem. Formally, for an input x and its compressed representation $\hat{z}$, which the server decodes to continue inference, one seeks to minimize distortion $D = \mathbb{E}[d(x, \hat{x})]$ for some distortion measure $d(x, \hat{x})$ on the final output or task loss under a constraint on rate R (the number of bits used for $\hat{z}$). The rate-distortion function $R(D)$ gives the theoretical minimum achievable rate for a distortion D [Sha59]. In our context, the distortion corresponds to reconstruction error of the original input, directly affecting task performance. Minimizing rate for a given distortion is equivalent to minimizing the mutual information $I(X; \hat{Z})$ between the input and its compressed representation, subject to an accuracy constraint. Intuitively, $I(X; \hat{Z})$ quantifies how many bits of information about X are retained in $\hat{Z}$. The information bottleneck principle

is a related trade-off, seeking a representation $\hat{Z}$ that maximizes target information (e.g., labels), and minimizes $I(X;\hat{Z})$. At the R-D optimum, the mutual information becomes $I(X;\hat{Z}) = R$ [Sha48].

The VAE objective naturally enforces this R-D tradeoff [ADK+23]. In a VAE, the encoder produces an approximate posterior $q_\phi(z|x)$ with prior $p_\theta(z)$ on the latent, minimizing the evidence lower bound (ELBO):

$$\mathcal{L}_{\text{VAE}} = \mathbb{E}_{q(z|x)}[-\log p_\theta(x|z)] + \beta\, D_{\text{KL}}(q_\phi(z|x) \,\|\, p_\theta(z)). \tag{2.1}$$

Here, the first term is a reconstruction loss and the second term is a weighted KL-divergence acting as a rate penalty. In the feature compression context, $p_\theta(x|z)$ can be interpreted as a decoder which tries to reconstruct the input (or sufficient statistic for the task) from z, and the KL term encourages $q(z|x)$ to match a prior $p(z)$ that has limited capacity. For example, if $p(z)$ is a fixed uniform or isotropic distribution, the KL bounds the information passed into z. Ahuja *et al.* leverage this connection casting the R-D optimization problem as training an equivalent VAE with an appropriate loss function [ADK+23]. In their formulation, the rate R is estimated via a learned model $p_\theta(z)$ (a **neural rate-estimator**) and distortion D is given by a task loss, the objective takes the form $R + \lambda D$. By adjusting λ and moving along the R-D curve, different compression levels are achieved. Notably, they report improved R-D outcomes compared to heuristic compression. This perspective highlights that mutual information $I(X;Z)$ is upper-bounded by the KL term and is minimized in the compression process.

Vector quantization (VQ) is a classical compression technique where continuous vectors are mapped to discrete indices via a finite codebook. The seminal VQ-VAE model by van den Oord *et al.* [VV+17] introduced a discrete latent bottleneck into the VAE framework. Instead of a parametric continuous posterior $q(z|x)$, VQ-VAE uses an encoder that outputs a continuous latent $z_e(x)$ which is then quantized to the nearest code vector in a learned codebook $e = \{e_k \in \mathbb{R}^d : k = 1, \ldots, K\}$. The quantized latent index k (or equivalently the code vector e_k) is fed to the decoder.

Formally, the quantizer Q performs:

$$k(x) = \arg\min_{j \in \{1,\dots,K\}} \|z_e(x) - e_j\|_2, \tag{2.2}$$

$$z_q(x) = e_{k(x)}, \tag{2.3}$$

so that $z_q(x)$ is a discrete (from a finite set of K vectors) approximation of $z_e(x)$. The decoder then produces $\hat{x} = g_\theta(z_q(x))$. Because the quantization operation is non-differentiable, VQ-VAE training uses a straight-through estimator: during back-propagation, the gradient $\nabla_{z_e}\mathcal{L}$ is passed unmodified to ∇_{z_q} (treating $z_q \approx z_e$ for gradient purposes). Meanwhile, the codebook is learned using a separate loss bringing code vectors closer to encoder outputs. The overall VQ-VAE loss is:

$$\mathcal{L}_{\text{VQ-VAE}} = \underbrace{\|x - \hat{x}\|^2}_{\text{reconstruction}} + \underbrace{\|\operatorname{sg}[z_e(x)] - e_k\|_2^2}_{\text{codebook}} + \beta \underbrace{\|z_e(x) - \operatorname{sg}[e_k]\|_2^2}_{\text{commitment}} \tag{2.4}$$

where $k = k(x)$ is the chosen code, and $\operatorname{sg}[\cdot]$ denotes the 'stop-gradient' operator. The first term trains the encoder and decoder to reconstruct the input (and optimize any task-specific loss) from the quantized latent. The codebook loss moves the selected code vector e_k toward the encoder output $z_e(x)$ ($\operatorname{sg}[z_e]$ ensuring the encoder is not affected by this term). This update mechanism is analogous to the centroid update step in the k-means clustering algorithm. The commitment loss pushes the encoder output toward the chosen code vector to encourage the encoder to 'commit' to that code and not produce arbitrarily large values [VV+17]. β is a hyperparameter (often $\beta \approx 0.25$) balancing the encoder commitment. Importantly, VQ-VAE assumes a uniform prior over the discrete code z, which means it does not explicitly include a KL term in the loss (since if the prior $p(z)$ is uniform over K codes, then $D_{\text{KL}}(q(z|x)|p(z))$ is constant and equal to $\log K$). VQ-VAE maximizes reconstruction under a fixed bottleneck size K, rather than explicitly penalizing information usage, where the information bottleneck capacity is set by K (the maximum mutual information $I(X;Z) \leq \log_2 K$ bits). In other words, VQ-VAE operates at a point on the R-D curve determined by K and the learned code usage. If all codes are used equally, the code entropy $H(Z) = \log_2 K$ and the model utilizes full capacity. If code

usage is skewed, the effective rate is lower. This approach avoids the 'posterior collapse' issue of continuous VAEs by using discrete latents with a fixed prior [VV+17], and empirically yields rich representations (for images, speech, etc.), especially when paired with decoders or priors.

Recent work by Oh *et al.* [OLBJ25] introduces **SplitFC**, a complementary compression scheme that avoids learning latent priors altogether. SplitFC treats each feature column in the activation matrix independently and drops columns stochastically based on their variance. Columns with low variance are probabilistically masked, and remaining columns are quantized using non-uniform, per-column bit-widths optimized under a rate constraint. This adaptive approach exploits activation sparsity during training and provides a closed-form solution for dropout probabilities and bit allocation. It achieves strong empirical compression, up to 100× in some cases, while maintaining accuracy, and includes convergence guarantees under mild assumptions. Unlike methods based on learned codebooks or discrete latents, SplitFC is fully analytic which may complement R-D based methods like VQ-VAE. While these methods offer distinct advantages, they rely heavily on element-wise quantization, motivating our exploration of VQ-VAE in SplitNNs.

2.3 Vector-Quantization with Lookup-Free Quantization

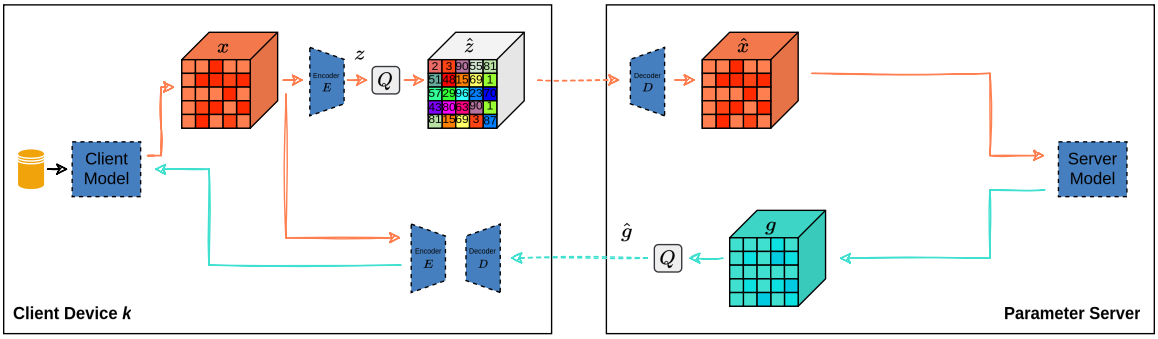

Figure 2.2: Overview of the proposed strategy illustrating the bottleneck and vector quantization components over a complete forward and backwards pass.

We introduce our compression method based on applying LFQ [YLG+24] within a VQ-VAE structure to the SplitNN setting. The edge-side encoder f_ϕ maps input x (activations at the split point) to a latent vector $z_e = f_\phi(x) \in \mathbb{R}^d$. Instead of transmitting z_e (which would be $32d$ bits if represented in f32), we compress it via lookup-free quantization [YLG+24]. The quantizer Q operates dimension-wise: each component $i \in \{1, \ldots, d\}$ is quantized independently to one of a small set S_i of values. For simplicity, assume each S_i contains M allowed values $\{v_{i,1}, \ldots, v_{i,M}\}$. The quantizer outputs $z_q = Q(z_e)$ defined by

$$[z_q]_i = \arg\min_{v \in S_i} |[z_e]_i - v| \tag{2.5}$$

i.e., choose the nearest allowed value for each coordinate. When $M = 2$ and $S_i = \{-1, 1\}$ for all i, this reduces to a 1-bit sign quantization example [YLG+24]. The collection of indices $(k_1, \ldots, k_d)$ such that $[z_q]_i = v_{i,k_i}$ is then the compressed representation. This product quantization scheme has an implicit codebook of size $K = M^d$ (cartesian product of all coordinate codewords), but crucially we never need to transmit or store a full codebook. Both encoder and decoder only need to know the small sets S_i. The decoder can directly take z_q as input, treating it just like a continuous latent. The decoder g_θ produces $\hat{x} = g_\theta(z_q)$, where $\hat{x}$ can be a reconstructed input or a feature/map that leading to the final task output. Because z_q is discrete, we can assign a short code to it for transmission: e.g., sending the indices $(k_1, \ldots, k_d)$, which costs $\sum_i \log_2 |S_i|$ bits (e.g., $d \log_2 M$ bits if each S_i has size M). d is chosen such that $d \log_2 M \ll$ size of original feature, achieving compression. In our split inference scenario, the edge device sends only the indices of z_q to the server, to then compute $\hat{x} = g_\theta(z_q)$. No other information is sent per inference. Periodically during training, the new decoder parameters θ and any updated quantization values S_i are communicated, which has negligible amortized cost compared to per-sample transmissions.

In a standard VQ-VAE, the codebook $e = \{e_k\}$ itself containing K d-dimensional vectors needs to be shared between the encoder side and decoder side. If the encoder outputs a code index k, the decoder must have the codebook to retrieve e_k. In a split network deployment, this is handled by ensuring both sides have synchronized copies of the model's parameters. LFQ takes this a step further by removing the

need for an arbitrary learned codebook. Instead, the quantization is defined by per-dimension rounding to fixed sets. This means that no embedding table is needed on either side, effectively, the 'codebook' is baked into the decoder's weights or a small set of quantization thresholds. In our case, rather than learning K separate embedding vectors, we learn a limited set of values for each latent dimension and use the Cartesian product of these values as the representation space [MMAT24]. The decoder can be implemented to decode these index combinations without a lookup table, for example, each quantized value can be fed directly into the first decoder layer. This property is advantageous in distributed settings: it reduces memory and removes a communication overhead that exists in other VQ approaches, especially if codebooks are frequently updated or specialized per client.

We train the encoder f_ϕ and decoder g_θ end-to-end to minimize a loss

$$\mathcal{L}(\phi, \theta, \psi) = \mathbb{E}_{(x,\hat{y})\sim\mathcal{D}}\Big[\ell_{task}\big(\hat{y}, h_\psi(\hat{x})\big) + \lambda\, \ell_{recon}\big(x, \hat{x}\big)\Big] + (\text{optional regularizers}) \quad (2.6)$$

where $\ell_{task}(x, \hat{x})$ is a task-specific loss. For reconstruction, ℓ_{recon} could be mean-squared error or cross-entropy, for classification via split features, ℓ_{task} could be a cross-entropy on the final predictions (meaning $g_\theta \circ Q \circ f_\phi$ plus the remaining server model are trained to predict labels). The quantization operation Q is nondifferentiable, so we adopt the VQ-VAE training tricks: treat $z_q = Q(z_e)$ as a constant during backprop and copy its gradient to z_e [VV+17]. We also use commitment losses to ensure the encoder's outputs do not drift far from quantized values. In our case, since each dimension's allowed values S_i are predetermined or slowly learned, one simple strategy is to clamp the encoder output via a penalty: $\sum_i \min_{v\in S_i} |[z_e]_i - v|^2$. This term encourages each $[z_e]_i$ to stay near one of the quantization grid points, preventing the encoder from producing values in between that would get quantized in an unstable way. This is analogous to the commitment loss in VQ-VAE. Additionally an entropy loss $\mathcal{L}_{\text{entropy}} = -\sum_{i=1}^{K} p_i \log p_i$ is included to encourage uniform usage of quantized codes, where p_i is the empirical code usage probability. A notable benefit of this approach is it sidesteps some of the complexities of codebook learning: because the 'codebook' is structural, we avoid codebook collapse and the need for tricks like code resets. Recent work on finite scalar quantization (FSQ) [MMAT24], similar

in concept to per-dimension quantization, shows it does not suffer from codebook collapse allowing the removal of complexity used in prior VQ methods (commitment losses, codebook reseeding, code splitting, entropy penalties, etc.) to learn discrete representations. In practice, we still include a small commitment loss to hasten convergence, but the training remains simplified, we train a continuous encoder/decoder with a rounding operation in between. This can be seen as training with quantization noise, a technique known to converge the network to a quantization-robust state [VV+17]. The convergence has similar guarantees to k-means clustering: quantization level updates seek to reduce reconstruction error for given encoder outputs (like k-means centroids minimizing distortion), while encoder/decoder updates seek to reduce loss for the given quantization. In VQ-VAE literature, the codebook update typically finds a good partition of the latent space [VV+17], and the encoder learns to produce outputs that hit those partitions, yielding a consistent assignment. Our scheme inherits this stability while reducing parameter overhead.

Each forward pass through the bottleneck yields a discrete latent z_q taking one of $K = \prod_i |S_i|$ possible values (combinations). As discussed for VQ-VAE, this implies an information capacity of at most $\log_2 K$ bits. The entropy $H(Z_q)$ of the latent (over the data distribution) represents the expected number of bits for transmission (rate R), and for a deterministic encoding $Z_q = f(X)$, $I(X; Z_q) = H(Z_q)$. Thus, the rate R is $H(Z_q) \leq \log_2 K$. In practice, to maximize efficiency, we want the encoder to use the discrete codes as uniformly as possible, which is encouraged by adding a small entropy regularizer: e.g., $-\gamma H(Z_q)$ to the loss, equivalent to penalizing $-\log p(Z_q)$, pushing the code usage toward uniform. In the original LFQ implementation [YLG+24] this entropy regularizer is introduced to encourage utilization of the code space. In our formulation, assuming a uniform prior over the discrete latent space (as in VQ-VAE, where this makes the KL term constant [VV+17]), the VAE perspective would consider the KL term as $\mathrm{KL}(q(z)||\mathrm{Uniform}) = \log K - H(Z_q)$. Not optimizing this term effectively fixes it to a constant related to $\log K$, whereas optimizing it (as in the Neural Rate Estimator method) would reduce $H(Z_q)$ at the cost of higher distortion. We can either set K just large enough to meet a desired distortion level (limiting $I(X; Z_q)$), or explicitly include a weighted $I(X; Z_q)$ term in the loss to enforce a

tradeoff. For analysis, one can imagine varying K (or adding an $I(X; Z_q)$ penalty) to trace out the empirical R-D curve of our model. We expect this approach to be competitive in its rate-distortion trade-off, especially as K grows large. In the high-rate regime, granular quantization allows D to approach zero (reconstruction nearly lossless) as R increases. In the low-rate regime, our approach gracefully degrades by using fewer effective codes. The distributed nature of the code (factorized across d dimensions) might not reach the absolute theoretical optimum for a given R because product quantization is a constrained form of vector quantization. However, product quantizers are known to be very effective in practice [MMAT24, YLG+24], and dramatically reduce complexity. While most previous demonstrations are in vision or audio, the principles apply equally to generic feature compression in distributed DNNs. Furthermore, in the distributed compression setting (each device compressing their own data), a universal codebook is beneficial.

2.4 Rate-Distortion Analysis

This section covers the rate-distortion properties of VQ-VAE + LFQ framework for compressing latent features $Z_e = f_\phi(X)$ from an input X. The encoder f_ϕ produces Z_e, which is quantized by Q to $Z_q = Q(Z_e)$. The decoder g_θ generates $\hat{X} = g_\theta(Z_q)$, used for downstream tasks. The distortion D measures fidelity loss, typically $D = \mathbb{E}[\|X - \hat{X}\|_2^2]$ for reconstruction, or a task-specific loss (e.g., cross-entropy) due to compression.

2.4.1 Rate-Distortion Relationships

The core trade-off between compression rate R (bits per sample) and distortion D is bounded by information-theoretic limits [Cov99, GG12]. For our VQ-VAE scheme with K effective quantization states (e.g., $K = M^d$ for LFQ with M levels across d dimensions), the following inequality chain holds:

$$R(D) \;\leq\; I(X; \hat{X}) \;\leq\; I(X; Z_q) \;=\; H(Z_q) \;\leq\; \log_2 K \tag{2.7}$$

Here, $R(D)$ is the rate-distortion function, the theoretical minimum rate for distortion D. $I(X; Z_q)$ is the mutual information between X and the quantized latent Z_q. Since Z_q is a deterministic function of X (given f_ϕ, Q), $H(Z_q|X) = 0$, thus $I(X; Z_q) = H(Z_q) - H(Z_q|X) = H(Z_q)$. The Shannon entropy $H(Z_q)$ of the quantized codes Z_q represents the achievable rate if optimal entropy coding (e.g., arithmetic coding) is applied to the transmitted indices. The maximum rate is $\log_2 K$ bits, occurring if all K states are used uniformly. Empirically, $H(Z_q)$ is often less than $\log_2 K$, for instance, with $K = 256$ ($\log_2 K = 8$ bits), observed entropy on a large dataset might be $H(Z_q) \approx 7.2$ bits. Our training may encourage uniform code usage (pushing $H(Z_q)$ towards $\log_2 K$) or learn a skewed distribution if it optimizes the overall R-D trade-off.

2.4.2 Quantization Performance and Limits

High-Rate Regime (Zador's Law) In the high-rate regime ($K \gg 1$ or rate $R \gg 0$), for a smooth d-dimensional source probability density function, Zador's theorem [Zad82, GG12] predicts that the distortion D for an optimal vector quantizer behaves as $D \approx C_d K^{-2/d}$ or, equivalently, $D \approx C'_d 2^{-2R_{avg}/d}$, where $R_{avg} \approx (\log_2 K)$ is the average rate. The constant C_d (or C'_d) depends on the source distribution and dimension d. For example, an IID. d-dimensional Gaussian source with variance σ^2 per dimension quantized using d independent scalar quantizers (rate R per dimension), $D \approx \frac{\sigma^2}{12} 2^{-2R}$ [Ger79]. While the LFQ scheme is a specific form of product quantization, these high-rate theories provide benchmarks for distortion decay.

Product Quantization (PQ) vs. Optimal Vector Quantization In the LFQ approach, quantizing each of the d latent dimensions $[Z_e]_i$ independently, is a product quantizer. PQ is computationally simpler than unstructured VQ but is optimal only if Z_e's dimensions are independent. If Z_e has correlated components, PQ can be suboptimal. However, the encoder f_ϕ can be trained to produce approximately decorrelated (factorized) latents Z_e, aligning them with LFQ's Cartesian grid structure and improving PQ efficiency. The impact of residual correlation ρ_{ij} between latent dimensions can affect performance. For instance, some analyses suggest that

if $\rho_{\max} = \max_{i \neq j} |\rho_{ij}|$, the distortion of a product quantizer D_{prod} might be bounded relative to an optimal full vector quantizer's distortion D_{full} by factors related to $\rho_{\max}$, e.g., $D_{\text{prod}} \leq (1 + \mathcal{O}(\rho_{\max}))D_{\text{full}}$ under certain conditions [GN98], underscoring the benefit of decorrelation. Empirically, Toderici et al. [TVJ+17] observed only a modest performance gap (e.g., ∼0.3 dB loss) for 1D PQ on CIFAR-10 image compression compared to more complex schemes. Training with a factorized prior, implicitly encouraged by LFQ, aids in learning such decorrelated features.

Lookup-Free Quantization (LFQ) Advantages LFQ [MMAT24, YLG+24] defines quantization levels structurally (e.g., fixed sets S_i per dimension), eliminating the need to learn, store, or transmit a large, arbitrary codebook. This structural nature helps avoid codebook collapse and simplifies training by potentially removing the need for complex VQ-VAE heuristics like codebook resets or strong commitment losses [MMAT24].

2.4.3 Training Dynamics and Convergence

The VQ-VAE with LFQ is trained end-to-end. Encoder f_ϕ and decoder g_θ parameters are updated via SGD, using the straight-through estimator (STE) to backpropagate gradients through the non-differentiable quantization $Z_q = Q(Z_e)$ [VV+17]. The training process resembles an alternating optimization:

1. Quantization $Z_q = Q(Z_e)$ maps Z_e to the nearest valid quantized values (assignment step).
2. The decoder g_θ is trained to minimize loss given Z_q.
3. The encoder f_ϕ is trained (via STE and often a commitment loss like $\|Z_e - \text{sg}(Z_q)\|_2^2$) to produce Z_e that minimizes overall loss after quantization.

This optimization is non-convex, so global optimality is not guaranteed. However, SGD typically guides the system towards a good local minimum where the empirical loss is non-increasing in expectation, analogous to k-means convergence [VV+17].

LFQ/FSQ also shows robustness to issues like 'dead codes' which can stabilize training [MMAT24].

2.4.4 Distributed Scenarios and Practical Benefits

In distributed settings like split inference or federated learning, our LFQ-based VQ-VAE offers:

Rate Adaptivity and Client Heterogeneity With non-IID client data $p_c(X)$, each client c uses the shared LFQ rule Q. The client-specific rate $R_c = H(Z_q^{(c)})$ (entropy of locally generated codes $Z_q^{(c)}$) adapts to $p_c(X)$. If $p_c(Z_q)$ is skewed, $R_c < \log_2 K$. The global decoder g_θ correctly interprets all Z_q. While a global LFQ is simple, its optimality for a client c can degrade if $p_c(Z_q)$ significantly diverges from the global average $p_{\text{global}}(Z_q)$ (potentially measurable by $D_{\text{KL}}(p_c(Z_q) \| p_{\text{global}}(Z_q))$). Lightweight per-client adaptations, like an affine scaling $Z'_e = a_c Z_e + b_c$ or temperature $Z'_e = Z_e / T_c$ pre-quantization, could improve client-specific performance with minimal overhead (parameters a_c, b_c or T_c).

Communication Efficiency A key advantage is drastically reduced bandwidth:

- **Low Per-Sample Payload**: Only indices for Z_q are sent. For L latent vectors per sample, each quantized by LFQ, the expected transmission size with optimal entropy coding is:

$$\mathbb{E}\big[\text{bits/sample}\big] \;\leq\; L \cdot H(Z_q) \;+\; O(1) \tag{2.8}$$

  where $H(Z_q)$ is the code entropy and $O(1)$ covers minor coding overheads [MMAT24, OLBJ25]. For example, if an intermediate feature tensor $16 \times 16 \times 256$ float32 activations of 262kB (typical in a ResNet) is compressed to $L = 1$ latent representation Z_q comprising $d = 256$ dimensions, each quantized to $M = 16$ levels (4 bits/dimension if uniform usage, i.e., $H(Z_q) \approx 4d = 1024$ bits), the payload becomes 128 bytes. This is a reduction of over $2000\times$. If $H(Z_q)$ is lower due to data statistics or fewer levels (e.g., $M = 2$, 1 bit/dim),

the reduction can be even more substantial (e.g., to 32 bytes, an 8000× reduction).

- **No Frequent Synchronization**: LFQ's structural 'codebook' eliminates the need for frequent codebook transmission or synchronization, unlike traditional VQ systems. Only model parameters (e.g., decoder weights θ) are sent during retraining rounds.

Scalability and Robustness Since the LFQ rule is common, it simplifies deployment across many clients. Each client can operate at its own effective rate R_c, providing adaptability, akin to [OLBJ25] but achieved via learned code usage over a fixed quantization scheme. This makes the system robust to diverse client data.

2.5 Evaluation

This section presents an empirical evaluation of the proposed VQ-VAE + LFQ. The primary goal is to assess the method's effectiveness in reducing communication overhead, measured by Bits Per element (BPP) of the feature vector and the resulting compression ratio relative to uncompressed 32-bit floating-point features, while maintaining high task performance (accuracy for classification, mean Intersection over Union (mIoU) for segmentation). We compare LFQ against standard Split Learning (Vanilla SL) and several established compression baselines across diverse datasets and network architectures, encompassing both non-IID split learning scenarios and IID split inference settings.

2.5.1 Experimental Setup

The evaluation employs standard benchmarks representative of typical split learning and inference tasks:

- **Non-IID Split Learning:** This setting simulates heterogeneous data distributions across clients which mimic environments that employ split learning

[GKA+20]. Client/Server splits and training configurations (SGD optimizer, learning rates, communication rounds) mirror setups like SplitFC [OLBJ25]. Baselines include Vanilla SL (no compression), FedLite, Top-S, and SplitFC [OLBJ25].

MNIST [LBBH98]: We use a LeNet-5 variant split after the second layer (client: 4,800 parameters, server: 148,874 parameters). Training uses SGD with an initial learning rate of 0.01, momentum of 0.9, and weight decay of 10^{-4}. The dataset is split into non-IID subsets across $K = 30$ clients, where each client is assigned exactly two digit labels [LSWX22]. Training runs for $T = 200$ communication rounds. The base feature dimension for compression ratio calculation is $\bar{D} = 1,152$ with a batch size $B = 256$.

CIFAR-10 [KH+09]: A VGG-16 model [SZ15] pre-trained on ImageNet-1k [DDS+09] is split at the 19th layer (client: 2,915,648 parameters, server: 131,385,866 parameters). Training uses SGD with an initial learning rate of 0.01, momentum of 0.9, and weight decay of 10^{-4}. Non-IID data distribution across $K = 20$ clients is generated using a Dirichlet distribution with concentration parameter $\beta = 0.3$ [LSWX22]. Training runs for $T = 200$ communication rounds. The base feature dimension is $\bar{D} = 8,192$ with a batch size $B = 256$.

CelebA [LLWT18]: A MobileNetV3-Large model [HSC+19] pre-trained on ImageNet-1k [DDS+09] is split after the 7th InvertedResidual block (client: 93,200 parameters, server: 4,111,394 parameters). Training uses SGD with an initial learning rate of 0.01, momentum of 0.9, and weight decay of 10^{-4}. The dataset is split across $K = 100$ clients uniquely by identity [JWV+23]. Training runs for $T = 100$ communication rounds. The base feature dimension is $\bar{D} = 13,400$ with a batch size $B = 64$.

- **IID Split Inference:** This setup assumes a pre-trained model trained as a single unit, later split for inference on edge devices processing IID data. Baselines include learned methods like Neural Rate Estimator (NRE) [ADK+23], Entropic Student [ML21], Variational methods [BMS+18], and standard codecs

like JPEG and HEVC.

Imagenet-1k [DDS+09]: A ResNet50 model [HZRS16] pretrained on ImageNet-1k is split after the 8th layer (split 1 in Ahuja *et al.*), with a resolution of 224×224. Training uses SGD with an initial learning rate of 0.01, momentum of 0.9, and weight decay of 10^{-4}. The number of epochs is $E = 15$ with a batch size $B = 128$.

MS-COCO 2017 [LMB+14]: A DeepLabv3 model [CPSA17] with the ResNet component pretrained on ImageNet-1k is split after 8th layer (split 2 in Ahuja *et al.*), with a resolution of 513×513. Training uses SGD with an initial learning rate of 0.003, momentum of 0.9, and weight decay of 10^{-4}. The number of epochs is $E = 10$ with a batch size $B = 40$.

The VQ-VAE encoder and decoder are implemented as simple ConvNets. Spatial dimensions are progressively downsampled using strided convolutions in the encoder and upsampled using transposed convolutions in the decoder. To reduce parameter count and computational complexity, we use depthwise separable convolutional topology [HZC+17] throughout. Each convolutional layer is followed by ReLU activation and batch normalization. We use $M = 2$ levels with $S_i = \{-1, 1\}$ for all LFQ experiments.

Metrics reported are top-1 accuracy for classification, mIoU for segmentation, BPP for the compressed features, and the compression ratio (calculated as uncompressed/compressed bits). Lower BPP and higher compression ratios indicate better communication efficiency.

2.5.2 Split Learning Results (Non-IID Data)

The performance of LFQ under non-IID conditions typical of federated or decentralized learning is presented below.

CIFAR-10 (VGG-16) On the CIFAR-10 dataset (Table 2.2, Figure 2.3), the uncompressed Vanilla SL baseline achieves 86.88% accuracy at 32 BPP. SplitFC maintains high accuracy (e.g., 86.09%) at significantly reduced rates (3.2 BPP, 80x ratio),

Table 2.2: CIFAR-10, Dirichlet beta=0.3, devices=20, VGG16 (cut layer=19), B=256, T=200

Method	BPP ↓	Accuracy ↑	Ratio ↑
Vanilla SL	32.0	86.88	1.0
FedLite	0.8	75.56	320.00
FedLite	1.6	80.57	160.00
FedLite	3.2	85.04	80.00
Top-S	0.8	79.71	320.00
Top-S	1.6	82.01	160.00
Top-S	3.2	83.16	80.00
SplitFC	0.8	85.31	320.00
SplitFC	1.6	85.98	160.00
SplitFC	3.2	86.09	80.00
LFQ [1 group]	0.051	79.56	2422.59
LFQ [2 group]	0.117	81.68	1516.24
LFQ [4 group]	0.202	82.60	830.30
LFQ (pre) [1 group]	0.051	83.04	2414.45
LFQ (pre) [2 group]	0.102	85.36	1669.68
LFQ (pre) [4 group]	0.203	86.19	1006.23

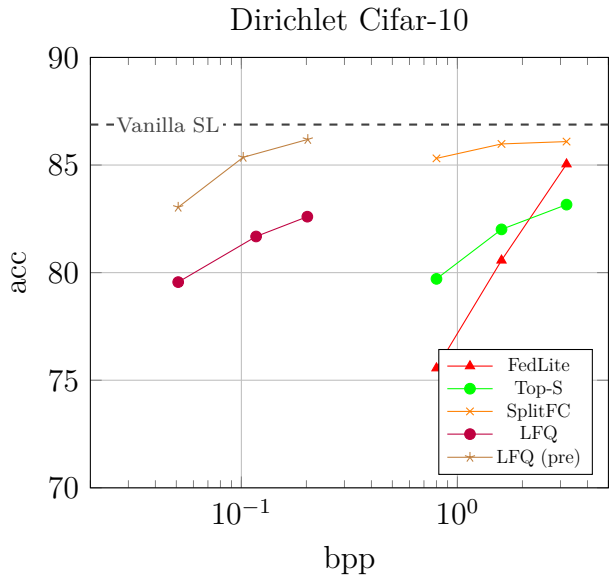

Figure 2.3: CIFAR-10, Dirichlet beta=0.3, devices=20, VGG16 (cut layer=19), B=256, T=200

while FedLite and Top-S experience noticeable accuracy degradation at comparable compression levels. Our proposed LFQ method achieves substantially lower bitrates, operating below 0.21 BPP. The standard LFQ variants show a trade-off, reaching 82.60% accuracy at 0.202 BPP (830x ratio). Notably, the LFQ variant using pre-trained components (`LFQ (pre)`) demonstrates strong performance, achieving 85.36% accuracy at only 0.102 BPP (1670x ratio) and 86.19% accuracy at 0.203 BPP (1006x ratio). This indicates that LFQ, especially with appropriate initialization or pre-training, can offer compression ratios an order of magnitude higher than SplitFC while maintaining competitive accuracy.

CelebA (MobileNetV3-Large) On the CelebA dataset (Table 2.3, Figure 2.4), Vanilla SL reaches 91.90% accuracy. SplitFC performs remarkably well, sustaining accuracy around 92.2% even at 0.2 BPP (320x ratio). FedLite and Top-S show significant accuracy drops at these compression levels. LFQ operates at even lower bitrates, ranging from 0.011 to 0.046 BPP. In this extremely low bitrate regime, LFQ maintains high accuracy levels (e.g., 92.37% at 0.046 BPP, 91.63% at 0.012 BPP),

Table 2.3: CelebA, even labels/partition, devices=100, MobileNetV3-large (cut layer =8), B=64, T=100

Method	BPP ↓	Accuracy ↑	Ratio ↑
Vanilla SL	32.0	91.90	1.0
FedLite	0.2	70.50	320.00
FedLite	0.4	82.44	160.00
FedLite	0.8	86.63	80.00
Top-S	0.2	53.02	320.00
Top-S	0.4	56.76	160.00
Top-S	0.8	62.18	80.00
SplitFC	0.2	91.84	320.00
SplitFC	0.4	92.22	160.00
SplitFC	0.8	92.23	80.00
LFQ [2 freq, 1 stride]	0.046	92.37	262.94
LFQ [4 freq, 1 stride]	0.046	92.17	274.83
LFQ [4 freq]	0.012	91.63	1031.26
LFQ [4 freq]	0.011	90.52	1132.26

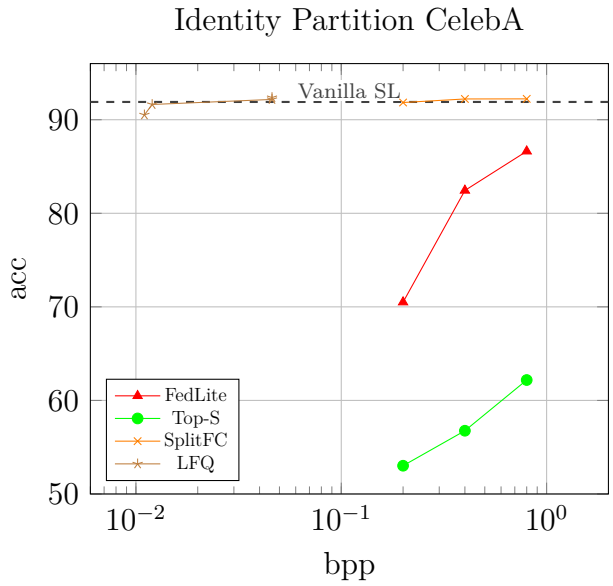

Figure 2.4: CelebA, even labels/partition, devices=100, MobileNetV3-large (cut layer =8), B=64, T=100

often matching or slightly exceeding the Vanilla SL baseline and significantly outperforming FedLite and Top-S. Compared to SplitFC, LFQ provides substantially higher compression ratios (over 1000x in some configurations) while preserving near-baseline task performance.

MNIST (LeNet-5) For the MNIST dataset (Table 2.4, Figure 2.5), the baseline accuracy is 98.93%. SplitFC again shows strong results, achieving approximately 97.8% accuracy down to 0.294 BPP (160x ratio). FedLite and Top-S exhibit larger performance drops. LFQ achieves very low bitrates around 0.1 BPP. At this rate, LFQ yields accuracy around 96%, which is slightly below the uncompressed baseline and SplitFC's best results, but represents a much higher compression ratio (∼450x) and significantly outperforms FedLite and Top-S in terms of accuracy preservation at low bitrates.

Overall (Non-IID) Across these non-IID benchmarks, LFQ consistently demonstrates the capability to operate at significantly lower bitrates (and thus higher com-

Table 2.4: MNIST, 2 labels/partition, devices=30, LeNet-5 (cut layer=2), B=256, T=200

Method	BPP ↓	Accuracy ↑	Ratio ↑
Vanilla SL	32.0	98.93	1.0
FedLite	0.147	70.19	320.00
FedLite	0.294	85.25	160.00
FedLite	0.588	94.73	80.00
Top-S	0.147	78.28	320.00
Top-S	0.294	79.05	160.00
Top-S	0.588	89.62	80.00
SplitFC	0.147	95.96	320.00
SplitFC	0.294	97.77	160.00
SplitFC	0.588	97.78	80.00
LFQ [8 freq]	0.097	95.80	486.66
LFQ [1 freq]	0.107	96.05	438.73

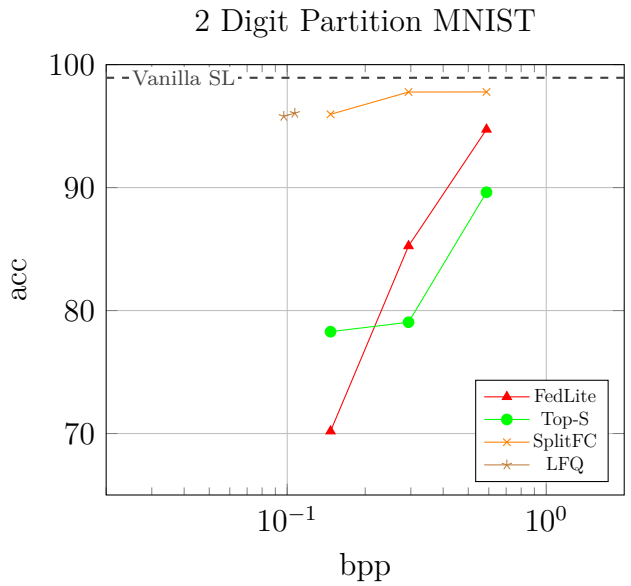

Figure 2.5: MNIST, 2 labels/partition, devices=30, LeNet-5 (cut layer=2), B=256, T=200

Table 2.5: Image segmentation showing the mIoU vs BPP tradeoff on COCO2017 using DeepLab v3. Our method achieves significantly lower bit-rates compared to other methods at similar mIoU levels.

Method	BPP ↓	MIoU ↑	Ratio ↑
Vanilla	32.0	66.4	1.00
LFQ	0.0080	62.17	131301.97
LFQ	0.0253	65.08	41583.24
LFQ	0.0620	66.16	16988.22
LFQ	0.0166	65.10	63584.17
LFQ	0.0166	64.86	63613.45

pression ratios) compared to the evaluated baselines. While a minor accuracy trade-off may exist compared to uncompressed transmission or the best performing SplitFC variants, LFQ maintains competitive accuracy levels, particularly exceeding other compression techniques like FedLite and Top-S, especially in the very low BPP regime.

2.5.3 Split Inference Results (IID Data)

In scenarios where a pre-trained model is split for inference, typically involving IID data streams, LFQ also shows substantial communication savings.

ImageNet Classification (ResNet50) The results for ImageNet classification are presented in Table 2.5 and implicitly compared against others in Figure 2.6. The uncompressed baseline achieves 66.4% accuracy. LFQ operates at extremely low bitrates, ranging from 0.008 BPP to 0.062 BPP. A clear rate-distortion trade-off is observed: accuracy increases from 62.17% at 0.008 BPP to 66.16% at 0.062 BPP. Notably, at 0.062 BPP, LFQ's accuracy is very close to the uncompressed baseline performance, while offering a compression ratio exceeding 500x (32 / 0.062). Figure 2.6 shows a rate-distortion curve, where LFQ shows a favorable position on the frontier, achieving competitive accuracy at substantially lower bitrates.

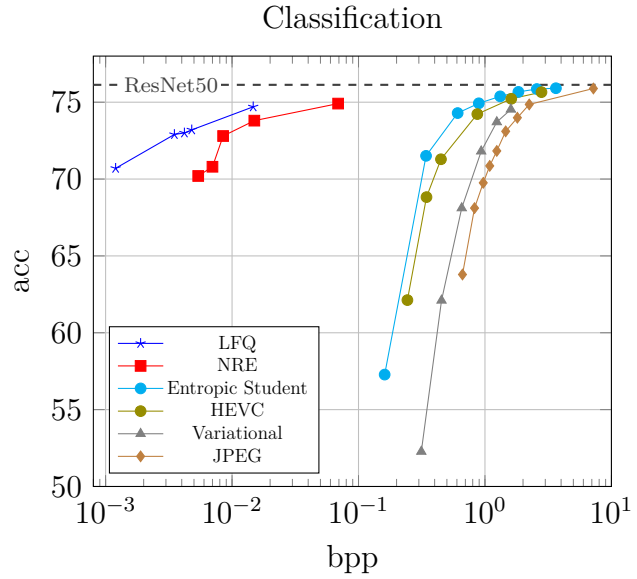

Figure 2.6: Image classification showing the accuracy vs BPP tradeoff on ImageNet-1k using ResNet50. Our method achieves significantly lower bit-rates compared to other methods at similar accuracy levels.

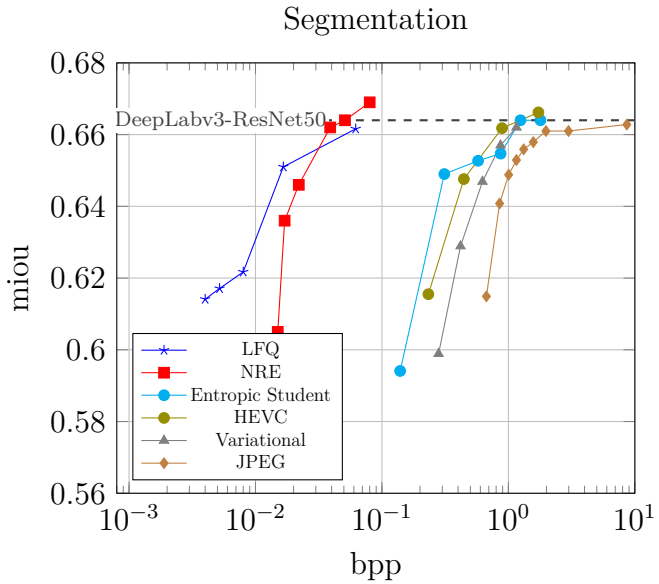

Figure 2.7: Image segmentation showing the mIOU vs BPP tradeoff on COCO2017 using Deeplab v3. Our method achieves significantly lower bit-rates compared to other methods at similar mIOU levels.

MS-COCO Segmentation (DeepLabv3) Figure 2.7 illustrates the performance on the MS-COCO segmentation task. The uncompressed DeepLabv3 baseline achieves an mIoU of approximately 0.664. LFQ results are clustered in the very low BPP region (0.004 to 0.062 BPP). Increasing the bitrate within this range improves performance, reaching an mIoU of 0.6616 at 0.062 BPP, nearly identical to the baseline. Compared to other methods shown in the plot (NRE, Entropic Student, HEVC, Variational, JPEG), LFQ achieves significantly better compression for any given mIoU level. For instance, reaching an mIoU close to 0.66 requires only $\sim$0.06 BPP with LFQ, whereas NRE requires $\sim$0.04 BPP (LFQ has slightly higher mIoU at 0.062 vs NRE at 0.039), and other methods like Entropic Student, HEVC, Variational, and JPEG require considerably higher bitrates (ranging from $\sim$0.9 BPP to over 2 BPP).

Overall (IID) In IID split inference tasks, LFQ demonstrates remarkable compression efficiency, achieving bitrates that are often orders of magnitude lower than traditional codecs and other learned compression techniques, while preserving task performance close to that of the original, uncompressed model.

2.5.4 Hyperparameter Sensitivity Analysis

Figure 2.8 presents a hyperparameter search for LFQ using ResNet18 on CIFAR-10, varying the network cut layer, VQ codebook size, number of quantization groups, number of quantizers per group, and the rate-distortion tradeoff parameter (λ).

- **Cut Layer:** Deeper cut layers (e.g., layers 7, 8) generally yield higher accuracy but result in significantly lower compression ratios (higher BPP), likely due to smaller but more complex feature maps at later stages.
- **Codebook Size, Groups, Num Quantizers:** Increasing these parameters tends to increase the bitrate (reduce compression ratio). Their impact on accuracy is varied: codebook size shows minimal effect in the tested range, increasing the number of quantizers offers slight accuracy gains up to a point, while increasing groups shows mixed effects depending on the cut layer. These results

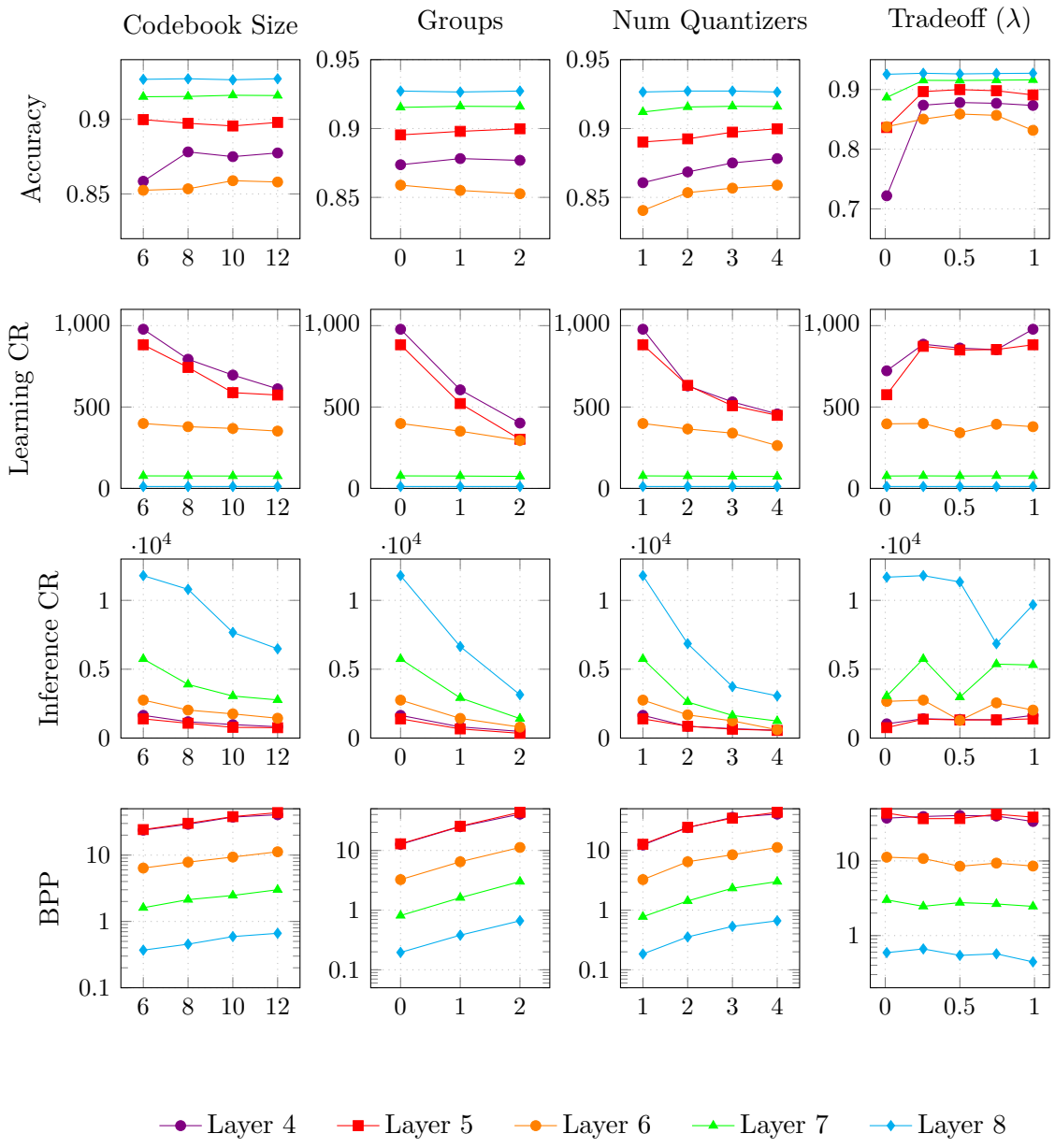

Figure 2.8: Hyperparameter search for Resnet18 on CIFAR-10 showing each variable vs. accuracy, compression ratio (w/ VQ decoder overhead), and compression ratio during inference. The plots are grouped by cut layer ranging from 4 to 8.

highlight the expected trade-offs between model complexity/capacity within the VQ module and the resulting rate-distortion performance.

- **Tradeoff (λ):** The tradeoff parameter λ behaves as anticipated, with intermediate values (e.g., 0.25–0.75) generally providing the best balance for accuracy, while higher values push towards lower bitrates, sometimes at the cost of accuracy.

This demonstrates that LFQ parameters influence the rate-distortion trade-off in predictable ways, allowing for tuning based on specific application requirements for accuracy versus communication cost.

2.5.5 Summary of Evaluation

The empirical results presented in this section validate the effectiveness of the proposed VQ-VAE + LFQ method for compressing intermediate features in split neural networks. Across various datasets (MNIST, CIFAR-10, CelebA, ImageNet, MS-COCO), network architectures, and data distributions (non-IID and IID), LFQ consistently achieves substantial reductions in communication cost (significantly lower BPP and higher compression ratios) compared to both uncompressed transmission and several baseline compression techniques, including SplitFC, FedLite, Top-S, NRE, standard codecs, and other learned methods (Tables 2.2, 2.3, 2.4, 2.5, Figure 2.7, 2.6). While operating at extremely low bitrates, LFQ generally maintains competitive task performance, often closely approaching the accuracy or mIoU of the uncompressed baseline models. The hyperparameter analysis further illustrates the controllable trade-offs offered by the method (Figure 2.8). These findings support the potential of VQ-VAE + LFQ as a highly communication-efficient approach for practical split learning and split inference deployments.

2.6 Discussion

The *Neural Rate Estimator* approach [ADK+23] integrates an explicit estimation of bit-rate into the training objective. In practice, they introduce a parametric model for $p(z)$ (often a hyperprior or an autoregressive density on the latent) so that $-\log_2 p(z)$ can serve as an estimate of code length. Their loss $L = \mathbb{E}[-\log_2 p_\theta(Z) - \log_2 p_\theta(Y|Z) + \lambda d(X, \hat{X})]$ combines an entropy term for the latent Z and a distortion term. By training the encoder to minimize this, they effectively minimize $I(X;Z) + \lambda D$. One advantage of that method is the ability to vary λ to trade off rate vs. distortion and achieve *variable bitrate* compression with one model. They also demonstrate an unsupervised training mode by using knowledge distillation as the distortion measure. Compared to our method, the neural rate-estimator uses continuous latents with entropy coding, whereas we use a fully discrete latent. In terms of communication overhead: their approach requires transmitting the compressed bitstream of z (analogous to our indices) *and* possibly the hyperprior bits if any. In contrast, our approach has no auxiliary bitstreams, the indices of z_q *are* the final code. Both approaches assume a model shared by edge and server. However, the neural rate-estimator's decoder must invert an entropy-coded representation, which means the server needs a probability model synchronized with the encoder to decode (e.g., an arithmetic decoder with the same $p(z)$). This is effectively a 'codebook' in a probabilistic sense. This makes our method slightly simpler in deployment, without need for custom encoding/decoding libraries. Performance-wise, both methods can achieve similar distortion at a given rate, however, our approach might use a bit more bits if the latent distribution is highly non-uniform. In scenarios where accurate rate control is needed, one could augment our method with a learned prior or simply adjust K or d. The simplicity of our method makes it appealing for real-time or hardware-constrained settings, there is no need to compute probabilities or sample from posteriors during inference, just a nearest-neighbor quantization.

The SplitFC framework [OLBJ25] targets the same split learning scenario but uses a different strategy: it identifies that different feature channels (columns in the feature matrix) have different 'dispersion' (variance and range), and thus not all

channels need equal precision. SplitFC applies two main steps per batch: (i) Adaptive Dropout: It probabilistically drops entire feature vectors (channels) that have low significance (measured by standard deviation), not transmitting them (and correspondingly dropping their gradients in backpropagation). (ii) Adaptive Quantization: For the remaining features, it chooses quantization levels based on each feature's value range, possibly using a two-stage quantizer (coarse and fine) or a mean-value quantizer focusing on the mean of the vector. They derive closed-form optimal quantization step sizes for a given number of bits to minimize error. Essentially, SplitFC allocates more bits to features with larger dynamic range and drops those with little variation. This approach is quite different from ours: it is data-dependent on a per-batch basis, whereas our method is data-independent (fixed quantizer) after training. SplitFC's strengths are that it can achieve additional compression by exploiting moment-to-moment redundancy (e.g., if some feature is nearly zero for all samples, it can be dropped entirely). In contrast, our VQ-VAE compression does not drop dimensions dynamically, always sending d quantized values. However, if a feature carries no information, our encoder can learn to output a constant value for it, which means the *entropy coder* would compress it to almost zero bits (since it is always the same symbol). So in a sense, some of that benefit is recovered at the coding stage (assuming we entropy-code the indices of z_q). From a theoretical viewpoint, the adaptive feature dropping is like a form of *variable-length coding* across features, which is powerful but complicates analysis (since the effective rate per sample is not fixed). Our method fixes the length but optimizes the content of that length. Another distinction is that SplitFC was demonstrated during *training* (each training iteration features are compressed and gradients returned), which focuses on not damaging the training convergence while saving communication. Our method is applicable to both training and inference: during training we treat $Q(f_\phi(x))$ as the forward pass and backpropagate through straight-through as usual. This yields the same effect of reduced gradient size (since gradients through dropped/quantized values can also be compressed or are sparse due to dropout). These ideas may be combined by using VQ quantization while also dropping any dimensions that stay near-zero (perhaps by setting $S_i = \{0\}$ for some i if the encoder learns it unnecessary). In terms of communication efficiency,

SplitFC achieved state-of-the-art results on benchmarks like CIFAR-100 and CelebA [OLBJ25], significantly reducing uplink and downlink costs. Our approach aims for a similar goal but with a discrete latent. We emphasize that transmitting only the decoder parameters periodically is a major advantage of our scheme. In SplitFC or others, one might need to communicate the chosen dropout mask or quantization levels for each batch from device to server (so that the server knows which features were dropped and how to interpret the quantized values). This overhead is not large (a few bytes per batch), but it introduces synchronization complexity. In our case, the server inherently knows how to decode every incoming z_q without any side information. The only time parameters are exchanged is when the model is updated (e.g.,new ϕ, θ after an epoch of federated averaging). At that time, the decoder's weights (and quantization value sets S_i if they were learned) are sent to devices. This is identical to standard model update costs and does not grow with the number of inferences. Thus, the communication pattern is simplified: edge sends compressed z_q, server sends nothing during inference (or sends back only the final prediction), and occasionally the server sends updated model parameters. This achieves minimal per-inference communication overhead.

Non-IID Data Distributions While the Neural Rate Estimator method has primarily evaluated on IID datasets (each client seeing a similar distribution as the whole), SplitFC has focused on non-IID datasets. The neural rate VAE method can handle non-IID in the sense that it does not assume any specific distribution, it learns a global model that tries to be robust. However, if one client's data produce very different intermediate feature distributions, the learned entropy model might assign suboptimal codes, increasing that client's bitrate. SplitFC's dropout strategy might shine in non-IID cases: for example, if a certain class of inputs (present only on client A) activates only a subset of features, those others can be dropped on that client, whereas on client B a different subset might be dropped. Their method can adapt per iteration, effectively customizing compression to the data in real-time. Our method, being more static, might in non-IID settings rely on the flexibility of the encoder network to accommodate different distributions. In a federated training scenario, one

could allow the encoder to have some client-specific parameters (a form of personalization) so that each learns to utilize the quantization levels optimally for its data. Even without that, the global model's discrete code space will be the union of what all clients needed to represent their features. In the worst case, if clients' feature distributions do not overlap at all, our codebook might need to be larger to serve all (increasing K). But if there is overlap or redundancy (which is often the case), a single codebook can efficiently represent all clients' features. In terms of accuracy, none of these compression methods should degrade the final performance if the bottleneck is trained properly at a sufficient rate. Ahuja *et al.* report improved task accuracy at lower bitrates compared to prior art [ADK+23], and SplitFC maintains accuracy while drastically cutting communication [OLBJ25]. We expect our method to likewise preserve accuracy as long as the latent code capacity K is adequate for the task complexity. We note that discrete representation learning has an additional benefit, which sometimes leads to improved generalization by forcing the model to learn more compact, symbolic features. This was observed in the original VQ-VAE, which discovered phoneme-like units in speech and semantic clusters in images [VV+17]. In split learning, a discrete bottleneck may provide a form of regularization that makes the model more robust to distribution shifts.

Decoder Transmission Efficient transmission of model components such as large decoder models from the client VQ-VAE to a server may be compressed to minimize communication overhead. Several well-studied model compression methods can significantly reduce the model size while retaining near-original inference accuracy in eval mode. Below, we note key techniques and recent advances in quantization [WMYY24], pruning [CZS24], low-rank factorization [CLL+23], knowledge distillation [GYMT21], self-compressing networks [CI23], and entropy coding, with an emphasis on maintaining decoder performance after compression.

- **Quantization:** Lowers the precision of weights (and possibly activations), for example from 32-bit float to 8-bit integer shrinking model size 4× and communication/storage needs [WMYY24]. Post-training quantization (**PTQ**) applies

this after training, offering speed but risking accuracy loss. Quantization-aware training (**QAT**) simulates quantization during training, generally achieving better accuracy, especially for lower bit-widths [WMYY24]. Both significantly reduce model update size.

- **Pruning:** Removes redundant parameters to create smaller models [CZS24]. Unstructured pruning zeros individual weights, resulting in sparse models that may require specialized hardware/libraries. Structured pruning removes entire neurons, filters, or channels, yielding smaller dense sub-networks easier to deploy and transmit. Pruning can remove a large fraction of weights (50-90%) with fine-tuning often recovering performance, reducing communication cost for updates.

- **Low-Rank Factorization:** Replaces large weight matrices/tensors with products of smaller ones using techniques like SVD or tensor decomposition (CP, Tucker) [CLL+23]. This reduces the parameter count by exploiting low-rank structure, especially in overparameterized layers. Applied post-training or integrated into training, it compresses the model by transmitting only the factors, useful for large decoders.

- **Knowledge Distillation:** Trains a smaller 'student' model to mimic the output behavior of a larger 'teacher' model [HVD15, GYMT21]. This transfers the learned function into a compact representation, allowing transmission of a significantly smaller student model (e.g., $>10\times$ size reduction reported [AAA21, GWC+25]) while preserving much of the teacher's performance.

- **Self-Compressing Networks:** A compression mechanism is (pruning, quantization) is introduced directly into the training objective (modifying the loss function) [CI23]. The network learns a compact structure inherently. For example, SCNNs simultaneously prune and quantize during training, producing highly compressed models ready for deployment (e.g., achieving high accuracy with $<20\%$ weights, $<5\%$ original bits [CI23]).

- **Entropy Coding and Extreme Bit-Reduction:** Entropy coding (e.g., DeepCABAC using adaptive arithmetic coding [WKM+19]) losslessly compresses quantized weights by statistical redundancy, achieving high ratios (e.g., 63× for VGG-16 [WKM+19]). Extreme quantization can push precision lower (e.g., BitNet using 1-bit weights [WMD+23]), which drastically reduces size while maintaining competitive performance.

- **Gradient Compression:** Distinct from model compression, gradient compression reduces communication during distributed training iterations [XHA+20]. Techniques include gradient sparsification (sending top-k gradients with error compensation) and gradient sketching (using randomized projections like Count Sketch [IRU+20]). These reduce per-iteration training costs, complementary to compressing the final model updates, such as in the SplitFed framework [TACS22].

2.7 Summary

Our use of VQ-VAE + LFQ provides empirical improvements to communication overhead seen in split neural networks. We show this follows a standard VQ formalization, while removing large codebooks communication by use of a lookup-free scheme, especially useful to our distributed setting. Instead, only the decoder weights need to be periodically transmitted during training. Compared to adaptive schemes, our method is *feed-forward deterministic* (no stochastic dropping), making it easier to analyze and implement. We optimize the bottleneck with a joint task and distortion objective end-to-end, ensuring that the learned compression is optimal for the task in question, rather than relying on heuristics per feature. Finally, we sidestep issues like posterior collapse, and KL-term complexity common to VAEs. By using a discrete bottleneck we inherit the simplicity of lookup-free methods. We show this approach offers empirically favorable results for compressing split network features in a discrete way to achieve high communication efficiency.

Chapter 3

Privacy Mechanisms in Split Learning

3.1 Introduction

Split Learning (SL) enables collaborative model training on resource-constrained devices by partitioning a neural network between a client and server. While offering baseline privacy by withholding raw data, the transmitted intermediate activations or 'smashed data' may remain vulnerable to inference and reconstruction attacks, exposing sensitive client information. Existing defenses, such as those based on adversarial training or information-theoretic regularization, often entangle privacy objectives with representation learning, potentially compromising task utility, or apply global transformations that lack fine-grained control. We propose PRISM (Privacy Router with Integrated Spatio-Channel Masking), a novel framework that addresses these limitations through an improved architecture, and information flow. PRISM utilizes a three-node U-shaped architecture (Client-In, Server, Client-Out) with a dynamic, learned masking module at the cut layer. This module performs fine-grained spatio-channel masking and routes information along two distinct paths: a heavily pruned, privacy-enhanced stream z_S sent to the server for computation, and a high-fidelity bypass stream z_C retained locally on the client. PRISM employs a disentangled optimization strategy: the client's primary feature encoder is optimized solely for

task performance using gradients from both streams, while the masking module is separately optimized to balance privacy and utility. Furthermore, PRISM incorporates explicit mask control mechanisms, including auxiliary loss terms, to regulate pruning behavior and prevent trivial solutions. This architectural and optimization decoupling allows PRISM to aggressively protect sensitive information in z_S without degrading the core representation quality, preserving utility through the local z_C path and offering tunable control over the privacy-utility trade-off. Empirical validation demonstrates superior privacy-utility trade-offs on CelebA and FairFace.

The proliferation of machine learning applications, particularly deep learning, across various domains has created immense demand for increasingly capable systems, but also significant challenges, especially concerning data privacy and computational resources. Split Learning (SL) [GR18] emerged as a promising distributed learning paradigm to address these challenges, particularly for scenarios involving edge devices or institutions with sensitive data (e.g., healthcare, finance). By partitioning a deep neural network architecture at a designated 'cut layer', SL allows a client (e.g., a mobile device or local hospital server) to perform initial computations locally on its private data x. The resulting intermediate activations z are then transmitted to a more powerful server, which executes the remaining network layers, computes the loss, and initiates backpropagation [GR18]. The client only receives gradients pertaining to its part of the model, thereby avoiding the need to share raw data directly.

This basic mechanism offers two primary advantages:

1. reduced computational burden on the client, enabling the use of larger, more complex models than feasible locally,

2. baseline privacy enhancement compared to centralized training, as raw data remains on the client's device.

However, the privacy protection afforded by vanilla SL is often insufficient and potentially misleading. Numerous studies have demonstrated that the intermediate activations z, despite being processed representations, can still contain substantial information about the original input x [GBDM20, ZH20, EKÇ22]. A curious or ma-

licious server, possessing z (and potentially gradients or model insights), can mount various attacks, including:

- **Reconstruction Attacks:** Attempting to reconstruct or approximate the original input x from z [EKÇ22].
- **Attribute Inference Attacks:** Inferring sensitive attributes s (e.g., demographics, identity, medical conditions) associated with x but potentially irrelevant to the primary task y [SS15, SRS17].
- **Property Inference Attacks:** Deducing properties of the client's dataset [MSDCS19].
- **Training-Time Attacks:** Malicious actors can also attempt to poison the training process itself, even under privacy-preserving schemes like differential privacy, to manipulate the final model's behavior [MZH19, ETÇ+24].

These vulnerabilities reinforce the need for stronger, active privacy-preserving mechanisms within the SL framework.

Existing approaches to enhance SL privacy broadly fall into several categories. Cryptographic methods (e.g., Homomorphic Encryption, Secure Multi-Party Computation) offer strong theoretical guarantees but often introduce significant computational and communication overhead [GDL+16]. Differential Privacy (DP) involves adding calibrated noise to activations or gradients to provide formal privacy guarantees [ACG+16, PTS+21], but typically comes at the cost of reduced model utility, especially in high-dimensional settings.

Consequently, much research has focused on learning-based defenses that aim to transform or filter the activations z to minimize leakage while preserving task utility. Adversarial Representation Learning (ARL) techniques train the client model to produce representations z that are simultaneously useful for the primary task and confusing for a proxy adversary trying to infer sensitive information [ES16, MMS+18, LGY+21]. Information-theoretic methods, like NoPeek [VSGR20], add regularization terms to the training objective to explicitly minimize statistical dependence (e.g.,

using distance correlation) between the input x and the activation z. More targeted approaches like DISCO [SCG+21] use dynamic channel pruning, learning an input-dependent filter to mask potentially sensitive channels in z before transmission, guided by an adversarial objective.

However, many existing learning-based methods suffer from limitations. ARL methods often rely on joint optimization, where a single loss function (balancing task utility and privacy penalty) influences both the primary feature extractor and the privacy mechanism. This entanglement can force the feature extractor to learn less informative representations to satisfy the privacy constraint, potentially degrading performance on the main task, especially when strong privacy is required [SCG+21]. Methods like NoPeek apply a global transformation or penalty, which might lack the granularity to selectively remove sensitive features while preserving task-relevant details, and might inadvertently blur useful information [VSGR20]. Furthermore, methods like DISCO discard the masked information entirely, losing any potential utility those features might have held.

In this work, we propose PRISM (Privacy Router with Integrated Spatio-Channel Masking), a novel framework designed to overcome these limitations through a combination of architectural modification and optimization strategy. PRISM introduces:

1. **A Three-Node U-Shaped Architecture with Bypass Stream:** Extending the U-shape concept originally used for label privacy [GR18], PRISM uses Client-In, Server, and Client-Out nodes. A dedicated masking module at the Client-In cut layer intelligently splits the intermediate representation z into two streams:

    - z_S: A privacy-enhanced stream, heavily pruned using learned spatio-channel masks, sent to the Server for computation offloading.
    - z_C: A high-fidelity bypass stream, containing the masked-out features, retained locally and routed directly to the Client-Out node.

2. **Integrated Spatio-Channel Masking:** PRISM's masking module operates at a finer granularity than channel-only methods, capable of masking specific

spatial locations within feature channels. This allows for more precise removal of localized sensitive information (e.g., a face region) while preserving useful features elsewhere in the same channel.

3. **Disentangled Optimization Strategy:** PRISM decouples the optimization objectives. The primary feature extractor (f_C^{in}) on the Client-In node is optimized purely based on the final task loss, receiving gradients propagated through both the server (z_S) path and the local bypass (z_C) path. This encourages it to learn rich representations for the task. The masking module (M_θ) is optimized separately using a dedicated objective which explicitly balances the utility derived from the server path against a privacy objective (e.g., an adversarial loss discouraging leakage from z_S).

4. **Explicit Mask Control:** Beyond the main privacy-utility trade-off parameter, PRISM incorporates auxiliary loss terms (L_{aux}) to directly regulate the behavior of the masking module, enforcing a target pruning ratio encouraging mask diversity, and preventing trivial solutions.

By actively *routing* information based on learned sensitivity and utility-sending less sensitive/computationally demanding features to the server (z_S) and keeping sensitive/complex features local (z_C), PRISM aims to achieve a superior privacy-utility trade-off. It minimizes leakage to the untrusted server by aggressively pruning z_S while preserving high task utility by leveraging the unpruned features in z_C within the trusted client environment via the Client-Out node. This section provides a comprehensive description of the PRISM framework, detailing its formulation and optimization, analyzes its privacy implications from an information-theoretic perspective, and contrasts it with prior work.

3.2 Background

PRISM builds upon and diverges from several lines of research in distributed learning, privacy-preserving machine learning, and representation learning. These

approaches can be roughly categorized into two types: structural modifications (e.g., pruning or architectural changes) or by learning-based regularization (e.g., adversarial objectives or information bottlenecks).

3.2.1 Split Learning and its Variants

Introduced by Gupta *et al.* [GR18], vanilla split learning (SL) involves a simple linear topology (Client → Server). Its primary benefits are computational offloading and avoiding raw data sharing. While vanilla SL provides a degree of opaqueness to intermediates values, an honest-but-curious server or eavesdropper can often reconstruct the input or infer private attributes from the shared activation [GBDM20, ZH20, EKÇ22]. For example, Fredrikson *et al.* [FJR15] showed that an attacker with access to model outputs can invert a model to recover images, and similarly, Dosovitskiy & Brox [DB16] demonstrated that intermediate CNN features can be inverted to recognizable images. Beyond the risks from exposed activations, vanilla SL also typically requires the server to access labels for loss computation, raising concerns if labels themselves are sensitive. Recognizing the label leakage issue, the U-shaped architecture was proposed [GR18], where the final layers and loss computation occur back on the client side (Client → Server → Client topology). PRISM adopts and repurposes this U-shaped structure, not for label privacy (though it inherently supports it), but as a mechanism for routing features based on input privacy concerns. Other SL variants focus on reducing communication [TACS22] or handling multiple clients [AKK+20]. However, the fundamental vulnerability of information leakage from the smashed data z persists across most variants unless actively addressed [GBDM20, EKÇ22].

3.2.2 Adversarial Representation Learning for Privacy

Adversarial Representation Learning (ARL) techniques adapt the concept of adversarial training [GPAM+20, MMS+18] for privacy preservation, or bias reduction [AZN19, WQK+20]. The goal is to learn representations that are invariant or non-informative for specific sensitive attributes s while remaining useful for the primary

task y. This is typically framed as a minimax game: the main model (encoder) minimizes task loss and maximizes the loss of a concurrently trained adversary that tries to predict s from the representation [ES16, GUA+17, KKK+19]. Methods like DeepObfuscator [LGY+21] apply this directly to intermediate representations in model inference. Roy and Boddeti [RB19] introduce a maximum-entropy adversarial loss, forcing intermediate representation to be as uninformative as possible. PrivacyNet [MRR20] take a semi-adversarial approach, training a GAN-based filter to perturb face images, hiding attributes like race or age while preserving the identity information for recognition. The common thread across these methods is the use of an auxiliary (adversary) network trained to penalize sensitive features.

While conceptually appealing, ARL approaches face challenges:

- **Training Instability:** Minimax optimization can be notoriously unstable and difficult to converge.
- **Residual Leakage:** The adversary is only a proxy for real threats. Information useful to a different or more powerful adversary might still remain in the representation [SR20].
- **Entanglement of Objectives:** As the encoder is penalized for encoding sensitive information, it might be forced to discard features that are correlated with both the sensitive attribute s and the task label y, leading to an unavoidable utility loss, especially if s and y are not independent [ES16, MMS+18]. The encoder learns a single representation, with *no structural change*, trying to satisfy competing goals.
- **Opacity:** The resulting representation is often a dense embedding, making it hard to interpret what information was removed or why it is considered private.

PRISM utilizes an adversarial objective to guide its masking module, but disentangles this objective from the training of the primary encoder and provides an alternative path (z_C) for potentially sensitive but useful information, aiming to mitigate the utility degradation issue.

3.2.3 Information-Theoretic Privacy Measures

Recognizing the limitations of relying on specific adversaries, some approaches focus on minimizing statistical dependence between the sensitive source (e.g., raw input x or attribute s) and the shared representation z. NoPeek [VSGR20] pioneered the use of Distance Correlation (dCor) [SRB07] for this purpose in SL. dCor can capture non-linear dependencies, and minimizing `dCor(x, z)` via a regularization term encourages z to be statistically independent of x, hindering reconstruction attacks. Other works explore minimizing Mutual Information (MI), $I(x; z)$ or $I(s; z)$, often using variational bounds or other approximations due to the difficulty of estimating MI directly [AFDM17, POv+19].

These methods offer a more fundamental measure of information leakage than reliance on a specific adversary. However, they typically operate by applying a global penalty or transformation to the entire representation z. This might lack the fine-grained control to selectively target only the sensitive parts of the representation. If sensitive information is spatially localized or within specific feature types, a global penalty might degrade other useful features to achieve the desired overall statistical independence. Furthermore, like ARL, they modify the single representation z sent to the server, potentially impacting utility if the required level of independence demands significant information removal. PRISM, while described with an adversarial objective, could potentially incorporate such metrics to guide its masker, but its key distinction lies in the architectural routing (bypass stream z_C) and the explicit, fine-grained masking mechanism, rather than solely relying on implicit regularization of the shared representation z_S.

3.2.4 Feature Pruning and Masking for Privacy

Techniques like network pruning, originally developed for model compression and efficiency [LDS89, HMD16], have been adapted for privacy. The most basic form is static feature suppression, where the client *a priori* remove or obfuscate certain parts of the activation. For instance dropping a subset of channels likely to carry sensitive information. These static defenses do not adapt to individual data samples,

and the masking pattern is *fixed* during training and inference. Another vain is the **information bottleneck**, or privacy-by-bottleneck approach. Osia *et al.* [OSS+20] is an example of this, sending intentionally compacted representations to the server through *Siamese fine-tuning*. This ensures the client's output contains no extra information except what is necessary for the main task, and can be seen as removing sensitive variance. In general, static suppression methods tend to have **low computation and memory overhead**, but are not adaptive to new kinds of leakage.

DISCO [SCG+21] is a notable improvement, and example of adaptive pruning. They introduce a dynamic, input-dependent filter on the client that learns to prune (set to zero) entire channels of the intermediate activation z. Removal is based on their correlation with sensitive attributes, guided by a joint adversarial objective similar to ARL ($\mathcal{L}_{\text{joint}} = \rho\mathcal{L}_{\text{task}} - (1-\rho)\mathcal{L}_{\text{adv}}$). The pruned activation z_{pruned} is then sent to the server.

DISCO demonstrates significant improvements over static methods and baseline SL. However, it shares the joint optimization drawback with ARL, potentially degrading the feature extractor's quality. More fundamentally:

- **Channel-Level Granularity:** Masking entire channels is coarse. If sensitive information is spatially localized (e.g., a face in a corner of an image), DISCO might unnecessarily discard the entire channel detecting that feature type, losing information from non-sensitive spatial regions. This is addressed by their *spatial decoupler* by patchifying the image, and rearranging into the channel dimension. However, this pre-processing step incurs a non-trivial compute and memory overhead.

- **Information Discarding:** Features (channels) masked by DISCO are entirely discarded from the computation. If a masked channel contained features useful for the task (even if also sensitive), that utility is permanently lost.

PRISM is directly inspired by DISCO but extends it significantly by introducing (1) finer-grained spatio-channel masking and (2) the bypass stream (z_C) to reroute rather than discard masked information, coupled with (3) the disentangled optimization strategy to protect the primary encoder.

PRISM additionally eliminates the pre-processing pipeline required by DISCO to prune spatial information. DISCO's approach applies a *spatial decoupler*: this involves reshaping feature maps into d^2 patches and applying scaled masking to sensitive channels. By removing these steps, PRISM improves training and inference speeds. In practice, this leads to roughly a 3× speedup on CelebA and FairFace, while consuming less memory, by not expanding the batch size (or channel dimension) of the first Conv layer. This gain comes without sacrificing privacy or accuracy, since spatial information is pruned later by the information router. By removing the preprocessing mechanism, PRISM has a more lightweight pipeline, which is useful especially on high-resolution datasets.

3.2.5 Comparison of Approaches

Table 3.1 provides a summary of the major split learning privacy mechanisms discussed, comparing their computational cost, memory overhead, training complexity, required architectural changes, adaptability to different tasks, type of privacy technique, and qualitative privacy strength. **PRISM** is highlighted as advancing the state-of-the-art by achieving *High* privacy strength with low overhead via a structural approach, whereas earlier methods typically trade off one aspect for another.

In the table, "Arch." refers to whether the base neural network must be modi-

Table 3.1: Comparison of privacy mechanisms in split learning. Privacy Strength indicates the effectiveness in preventing leakage. Icons: 🟢 Low Overhead/Complexity or High Strength, 🟡 Moderate Overhead/Complexity or Strength, 🔴 High Overhead/Complexity or Low Strength.

Method	Compute Overhead	Memory Overhead	Complexity	Arch. Change	Adaptability	Privacy Strength
Vanilla	🟢	🟢	🟢	No	–	🔴
Static	🟢	🟢	🟢	No	Limited	🔴
NoPeek [VSGR20]	🟡	🟢	🟡	No	General	🟡
ARL [KKK+19]	🟡	🟡	🔴	No	Targeted	🟡
Pruning [SCG+21]	🔴	🔴	🔴	Yes	General	🟢
U-Shape (PRISM)	🟡	🟡	🔴	Yes	General	🟢

fied (beyond inserting a new loss or small module). For example, DISCO and PRISM introduce new modules (thus "Yes"), whereas ARL and NoPeek operate on an unmodified architecture. "Adaptability" indicates whether the method can easily generalize to protect different types of information: "Targeted" means one must retrain or adjust the defense for each specific sensitive attribute (as in ARL, which typically assumes a known attribute to protect), while "General" means the approach inherently protects against a broad range of leakages (e.g., NoPeek, DISCO and PRISM aim to hide any information not relevant to the primary task). Dynamic pruning methods like DISCO achieve a favorable profile with moderate overhead, and strong privacy. By contrast, adversarial methods, while offering moderate privacy, incur high training complexity, and static methods, though cheap, provide only limited protection. Finally, U-shaped architectures like PRISM aim to combine privacy benefits with manageable overheads through structural design.

3.3 Methodology

PRISM adopts a U-shaped split learning structure, introducing a dynamic routing mechanism based on spatio-channel masking, and a disentangled optimization strategy to keep sensitive info locally contained, while preserving utility in split learning.

3.3.1 System Architecture: Three-Node U-Shape with Bypass

PRISM utilizes a three-node U-shaped configuration, repurposing the topology for feature routing and privacy control (Figure 3.1):

1. **Client-In Node:** Resides on the client's trusted device (e.g., edge device, local server). It hosts:
    - The initial segment of the neural network, the encoder f_C^{in}, parameterized by θ_{in}. It processes the raw input x.

- The dynamic masking and routing module M_θ, parameterized by θ_M. It takes the output of f_C^{in} and determines the routing path for each feature element.

2. **Server Node:** Resides on a potentially untrusted, computationally powerful server. It hosts:
   - The intermediate segment of the network, f_S, parameterized by θ_S. It processes only the pruned, privacy-enhanced features received from the Client-In node.
3. **Client-Out Node:** Resides on the client's trusted device (can be the same physical device as Client-In). It hosts:
   - The final segment of the task model, f_C^{out}, parameterized by θ_{out}. This includes mechanisms for fusing information from the server and the local bypass stream, and the final layers (e.g., classifier head).

This distributed architecture explicitly separates the functions of initial feature extraction (f_C^{in}), privacy-critical filtering and routing (M_θ), computationally intensive processing (f_S), and final prediction using combined information (f_C^{out}).

3.3.2 Mathematical Definitions

Let $x \in \mathcal{X}$ represent the input data sample, $y \in \mathcal{Y}$ the corresponding label for the primary task, and $s \in \mathcal{S}$ a sensitive attribute associated with x. The components are defined as:

- **Client-In Encoder:** $f_C^{in} : \mathcal{X} \to \mathbb{R}^{C \times H \times W}$. Maps input x to a full intermediate feature representation $z = f_C^{in}(x; \theta_{in})$. z is typically a 3D tensor (Channels, Height, Width).
- **Masking/Routing Module:** $M_\theta : \mathbb{R}^{C \times H \times W} \to \{0, 1\}^{C \times H \times W}$. Generates a binary mask $M = M_\theta(z; \theta_M)$ of the same shape as z, where $M_{ijk} \in \{0, 1\}$. Details in Section 3.3.3.

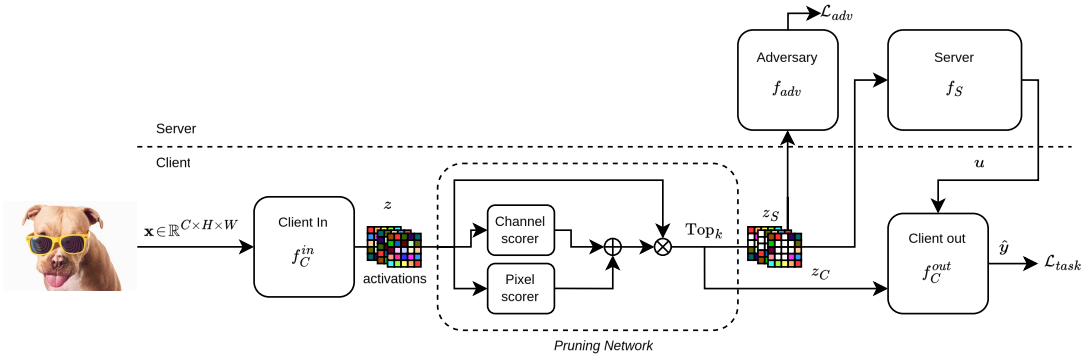

Figure 3.1: System overview of PRISM's U-shaped split network. Solid arrows show forward activations. The Client-In node computes $z = f_C^{in}(x)$ and uses the mask $M_\theta(z)$ to split it into z_S (sent to Server) and z_C (bypassed locally). The Server computes $u = f_S(z_S)$ and sends it to the Client-Out node. The Client-Out node fuses u with locally processed z_C to produce the final prediction $\hat{y}$.

- **Feature Splitting:** z is split element-wise using the mask:

$$z_S = M \odot z \qquad \text{(Server-bound features)} \tag{3.1}$$

$$z_C = (1 - M) \odot z \qquad \text{(Client-bypass features)} \tag{3.2}$$

  where $\odot$ denotes the Hadamard (element-wise) product. z_S contains features where $M = 1$, and z_C contains features where $M = 0$.

- **Server Model:** $f_S : \mathbb{R}^{C \times H \times W} \to \mathbb{R}^{C' \times H' \times W'}$. Processes the (sparse) pruned features z_S to produce intermediate server output $u = f_S(z_S; \theta_S)$. Note that f_S can change the dimensions.

- **Client-Out Model (including Fusion):** $f_C^{out} : (\mathbb{R}^{C' \times H' \times W'}, \mathbb{R}^{C'' \times H' \times W'}) \to \mathcal{Y}$. Receives u from the server and a locally processed version of z_C (denoted z'_C, potentially downsampled to match u's spatial dimensions, see Section 3.3.3), fuses them (Section 3.3.4), and produces the final prediction $\hat{y}$. The function f_C^{out} is parameterized by θ_{out}.

- **Overall Forward Pass Equation:**

$$\hat{y} = f_C^{out}(\underbrace{f_S(M \odot z; \theta_S)}_{u}, \underbrace{\text{Process}_{local}((1 - M) \odot z)}_{z'_C}; \theta_{out}), \tag{3.3}$$

where $z = f_C^{in}(x; \theta_{in}), M = M_\theta(z; \theta_M)$.

- **Adversary Model (for training):** $f_{adv} : \mathbb{R}^{C \times H \times W} \to \mathcal{S}$. A proxy adversary model, parameterized by θ_{adv}, trained alongside the main model. It attempts to predict the sensitive attribute s using *only* the server-bound features z_S: $\hat{s} = f_{adv}(z_S; \theta_{adv})$. This adversary provides the signal for the privacy component of the masker's objective.

3.3.3 Dynamic Information Routing (Spatio-Channel Masking Module)

The masking module M_θ is the core of PRISM's privacy mechanism and routing logic, implemented as a small convolutional network (e.g. 3x3 Conv layers, but may be swapped for attention mechanisms) that operates on z. It performs the following steps:

1. **Score Generation:** Given z, the module computes importance or sensitivity scores for each element (or groups of elements) in z. Different `scoring_style` strategies allow flexibility:

    - `channel`: Scores are generated per channel (e.g., using global average pooling within the mask network followed by a linear layer), resulting in a mask that is constant across spatial dimensions for a given channel. This mimics channel pruning.
    - `pixel` or `spatial`: Scores are generated per spatial location (e.g., using 1x1 convolutions to create a spatial attention map), potentially shared across channels or computed per channel. This allows masking specific regions.
    - `hybrid`: Combines channel and spatial scores (e.g., via element-wise multiplication or a learned linear combination using a `score_weight` parameter), allowing the module to adaptively balance between masking entire feature types and specific spatial regions.

2. **Binary Mask Generation (Differentiable Training):** The scores must be converted into a binary mask $M \in \{0,1\}^{C\times H\times W}$. Direct thresholding is non-differentiable. To enable gradient-based training of the masker parameters θ_M, we employ techniques used in learned pruning and gating:

   - **Thresholding:** A threshold τ determines the split. This threshold can be handled in various ways: `fixed` (a predefined value), `learned` (a trainable parameter), `statistical` (e.g., mean or median of scores), or `topk` (dynamically set to keep a fixed number or ratio $1 -$ `pruning_ratio` of scores above the threshold, ensuring a consistent masking budget).
   - **Continuous Relaxation:** Scores (centered and scaled, e.g., $(scores - \tau)/T$) are passed through a sigmoid function $\sigma(\cdot)$ or a similar smooth approximation (like the concrete distribution [MMT17]) to obtain soft mask values $M_{soft} \in [0,1]$. The temperature parameter T controls the steepness. It can be annealed during training to push soft masks towards 0 or 1 [SCG+21].
   - **Straight-Through Estimator (STE):** For the forward pass during training and at inference, a hard binary mask $M_{hard} = scores \geq \tau$ (or derived from `topk`) is used. During backpropagation, gradients are computed with respect to the soft mask M_{soft}, but are passed "straight through" the binarization step to update θ_M. Formally, $M = M_{hard}$ in forward pass, with $\frac{\partial \mathcal{L}}{\partial \theta_M} = \frac{\partial \mathcal{L}}{\partial M_{soft}} \frac{\partial M_{soft}}{\partial \theta_M}$ [BLC13]. This allows end-to-end training of the discrete mask generation process.

3. **Feature Splitting:** The hard binary mask M is applied element-wise: $z_S = M \odot z$ and $z_C = (1-M) \odot z$. Optionally, learnable 1x1 convolutions (`rescale_server`, `rescale_bypass`) can be applied to z_S and z_C immediately after masking to rescale features or adjust channel dimensions independently for each stream.

4. **Bypass Processing:** The bypassed features z_C remain on the client. Since the server model f_S might perform operations (like striding or pooling) that change the spatial dimensions of z_S to produce u, the Client-Out node needs

to align z_C with u before fusion. This is done by a local processing block, `BypassDownsample`, within f_C^{out} designed to match the spatial dimensions (H', W') and potentially the channel dimension (C'' vs C') of u. The output is z'_C.

This modular design isolates the privacy-filtering logic within M_θ. It allows f_C^{in} to focus on creating informative z, while M_θ learns the optimal routing strategy based on downstream signals (task performance via z_S and privacy penalty on z_S). The motivation for including auxiliary regularization terms ($\mathcal{L}_{aux}$ in Sec 3.3.5) during the training of M_θ is crucial to prevent trivial solutions (e.g., M_θ learning to always output all zeros or all ones, regardless of input or other losses) and enforce desirable properties like adhering to a specific average pruning ratio.

3.3.4 Feature Fusion at Client-Out

The Client-Out node f_C^{out} receives the processed features u from the server (derived from the privacy-filtered z_S) and the locally processed bypassed features z'_C. These two streams, representing computationally leveraged public information and high-fidelity private information respectively, must be fused effectively. PRISM supports several `fusion` strategies within f_C^{out}:

- **Residual Addition:** $z_{fused} = u + z'_C$. This is the simplest approach, assuming u and z'_C have compatible shapes (same C', H', W'). Requires careful design of f_S and `BypassDownsample` to ensure shape compatibility.
- **Concatenation:** $z_{combined} = \text{concat}([u, z'_C], \text{dim} = 1)$. Features are concatenated along the channel dimension. Typically, a subsequent convolution (e.g., 1x1 `fusion_conv`) is applied to reduce the channel dimension and mix the information: $z_{fused} = \text{fusion\_conv}(z_{combined})$. This is more flexible regarding channel dimensions C' and C''.
- **Hybrid Fusion:** Combines elements of both. For example,

$$z_{fused} = \text{fusion\_conv}(\text{concat}([u, z'_C], \text{dim} = 1)) + u.$$

This allows the model to learn a combination via the convolution while also potentially preserving the direct signal from the server path via the residual connection. This approach was found effective in initial implementations.

The choice of fusion mechanism impacts how the potentially noisy or less informative server stream interacts with the clean local stream, influencing final task performance and potentially robustness. This can alternatively be seen as decoding u using a reconstruction 'key' z'_C that the server never sees. The fused representation z_{fused} is then passed through the remaining layers of f_C^{out} (e.g., residual blocks, global pooling, final linear layer) to produce the prediction $\hat{y}$.

3.3.5 Optimization Strategy: Disentanglement and Control

PRISM adopts its disentangled optimization strategy, which treats the updates for the primary encoder (f_C^{in}) and the privacy masking module (M_θ) differently, from DISCO [SCG+21], with the addition of a gradient stream from the Client-Out model.

Loss Functions: The training involves multiple objectives:

1. **Task Loss ($\mathcal{L}_{\text{task}}$):** The primary objective for the main task. Computed at the Client-Out node based on the final prediction $\hat{y}$ and the ground truth label y.

$$\mathcal{L}_{\text{task}} = \mathbb{E}_{(x,y)\sim\mathcal{D}}[\ell_{\text{task}}(f_C^{out}(u, z'_C; \theta_{out}), y)] \tag{3.4}$$

   where ℓ_{task} is a suitable loss function (e.g., cross-entropy for classification) and (u, z'_C) are derived from x via Eq. (3.3).

2. **Adversarial Loss ($\mathcal{L}_{\text{adv}}$):** Measures the success of the proxy adversary f_{adv} in predicting the sensitive attribute s from the server-bound features z_S.

$$\mathcal{L}_{\text{adv}} = \mathbb{E}_{(x,s)\sim\mathcal{D}}[\ell_{\text{adv}}(f_{adv}(z_S; \theta_{adv}), s)] \tag{3.5}$$

   where ℓ_{adv} is the adversary's objective function (e.g., cross-entropy if s is categorical).

3. **Masking Module Loss ($\mathcal{L}_{\text{mask}}^{\text{total}}$):** This composite loss guides the parameters θ_M of the masking module M_θ. It consists of two main parts: a privacy-utility balancing term and auxiliary regularization terms.

$$\mathcal{L}_{\text{mask}}^{\text{total}} = \underbrace{\rho\mathcal{L}_{\text{task}}^{\text{server}} - (1-\rho)\mathcal{L}_{\text{adv}}^{\text{masker}}}_{\mathcal{L}_{\text{mask}}\text{:Privacy-Utility Balance}} + \underbrace{\gamma\mathcal{L}_{aux}}_{\text{Auxiliary Regularization}} \tag{3.6}$$

- $\mathcal{L}_{\text{task}}^{\text{server}}$: Represents the task utility signal derived only from the server path. It is the gradient of the main task loss $\mathcal{L}_{\text{task}}$ backpropagated through f_C^{out} and f_S up to z_S, and then considered as a loss signal for M_θ. Minimizing this term encourages the masker to preserve features in z_S that are useful for the task via the server computation.
- $\mathcal{L}_{\text{adv}}^{\text{masker}}$: Represents the privacy penalty signal. It is derived from the adversary's loss $\mathcal{L}_{\text{adv}}$, but calculated without detaching z_S, so gradients flow back into M_θ. Minimizing $-\mathcal{L}_{\text{adv}}^{\text{masker}}$ encourages the masker to produce z_S that confuses the adversary (maximizes the adversary's loss).
- $\rho \in [0, 1]$: A hyperparameter controlling the trade-off between utility preservation on the server path (high ρ) and privacy protection (low ρ) when updating the masker. $\rho = 0.5$ gives equal weight.
- $\mathcal{L}_{aux}$: Auxiliary loss terms that provide direct regularization on the mask itself, controlled by weight γ. These are crucial for stable training and meaningful masks. Common components include:
  - *Sparsity / Ratio Control Penalty:* Encourages the generated mask M to match a target sparsity level or `pruning_ratio`. For example, penalizing the deviation of the mean of M_{hard} (fraction of features sent to server) from a target ratio $1-r_{target}$: $(\text{mean}(M_{hard}) - (1-r_{target}))^2$. This prevents the mask from collapsing to all zeros or all ones and allows explicit control over the server load / bypass amount.
  - *Entropy Regularization:* Applied to the soft mask probabilities M_{soft} to encourage 0/1 values. E.g., minimizing the entropy $H(M_{soft})$.

Alternative Privacy Objectives for $\mathcal{L}_{\text{mask}}$: While the primary formulation above uses an adversarial loss $\mathcal{L}_{\text{adv}}$ as the privacy signal, the privacy term within $\mathcal{L}_{\text{mask}}$ could be replaced or augmented by information-theoretic regularizers, avoiding the need to train an explicit adversary f_{adv}:

- **Distance Correlation (dCor):** Inspired by NoPeek [VSGR20], one could define the privacy penalty as $\mathcal{L}_{priv} = dCor(z_S, x)$ or $dCor(z_S, s)$, aiming to minimize the statistical dependence between the server-bound features and the raw input or sensitive attribute. This requires efficient computation or approximation of dCor.
- **Mutual Information (MI):** The goal could be to directly minimize $I(z_S; x)$ or $I(z_S; s)$. As MI is hard to compute directly, variational bounds [AFDM17, POv+19] or alternative estimators could be used to approximate and minimize it.

Using such metrics might offer robustness against unforeseen adversaries but could be computationally more demanding or provide weaker signals compared to a targeted adversary. The choice depends on the specific application's threat model and computational constraints.

Disentangled Training Procedure: The parameters $(\theta_{in}, \theta_M, \theta_S, \theta_{out}, \theta_{adv})$ are updated iteratively as follows:

- **Main Task Model Update:** The client's encoder (f_C^{in}), the server's model (f_S), and the client's output model (f_C^{out}), are trained jointly to minimize the main task loss, $\mathcal{L}_{\text{task}}$. The encoder f_C^{in} receives task-related gradients from both the server path (via z_S) and the local bypass path (via z_C), encouraging it to learn utility-focused features.
- **Adversary Update:** Concurrently, the adversary f_{adv} is trained to minimize its prediction loss, $\mathcal{L}_{\text{adv}}$, using only the server-bound features z_S. To ensure the adversary's training only improves the adversary itself and does not influence

the encoder, gradients from $\mathcal{L}_{\text{adv}}$ are stopped from flowing back into f_C^{in} during this step.

- **Masking Module Update:** The masking module M_θ is uniquely optimized using a dedicated composite loss, $\mathcal{L}_{\text{mask}}^{\text{total}}$, which balances three objectives:

  1. **Utility Preservation:** It is encouraged to pass task-relevant features to the server (by minimizing the task loss signal propagated back to z_S).
  2. **Privacy Enhancement:** It is penalized for passing features that help the adversary (by maximizing $\mathcal{L}_{\text{adv}}$). This minimax objective can also be implemented efficiently using a Gradient Reversal Layer (GRL) [GUA+17].
  3. **Regularization:** It is guided by an auxiliary loss, $\mathcal{L}_{aux}$, to enforce desirable properties like a target sparsity ratio, preventing trivial solutions.

This training strategy is 'disentangled', optimizing the primary feature encoder f_C^{in} purely for task utility without being directly penalized by the adversarial loss. Instead, the specialized masking module M_θ is solely responsible for managing the privacy-utility trade-off on the information sent to the server, while the bypass path (z_C) provides a safety net for utility. This contrasts with joint optimization approaches where the encoder itself is penalized by the privacy objectives.

3.4 Evaluation

This section evaluates the performance of the proposed PRISM framework, focusing on its ability to balance primary task utility with privacy protection against inference and reconstruction attacks. We present quantitative and qualitative results on standard benchmark datasets, comparing PRISM against baseline Split Learning (Vanilla SL) and current state-of-the-art dynamic masking approach, DISCO [SCG+21]. The evaluation aims to empirically validate the benefits of PRISM's core components: the U-shaped architecture with bypass stream, spatio-channel masking, and disentangled optimization.

3.4.1 Experimental Setup

The evaluation utilizes three distinct datasets: *CelebA* [LLWT18], *FairFace* [KJ19], and *CIFAR-10* [KH+09]. For CelebA and FairFace, the primary task is typically facial attribute classification or identity recognition, while sensitive attributes involve demographic information (e.g., gender, race). For CIFAR-10, the task is object classification, and sensitive attribute is living.

Performance is measured along several axes:

- **Utility:** Accuracy on the primary classification task.
- **Privacy (Adversary Accuracy):** Accuracy of a trained adversary attempting to infer a sensitive attribute from the server-bound smashed data (z_S). Lower adversary accuracy indicates better privacy against attribute inference.
- **Privacy (Reconstruction Quality):** Metrics assessing the quality of images reconstructed from the smashed data (z_S) using a supervised decoder attack, mimicking an attacker trying to recover the original input. We use Peak Signal-to-Noise Ratio (PSNR) [HZ10], Structural Similarity Index Measure (SSIM), and Learned Perceptual Image Patch Similarity (LPIPS) [ZIE+18].

We compare PRISM against, **Vanilla SL** Standard SplitNN with no explicit privacy mechanism, and **DISCO** [SCG+21] A dynamic channel pruning method using joint optimization, with both pre-processing and without. PRISM is evaluated across a range of hyperparameter settings (denoted PRISM (X), where X corresponds to the target pruning ratio or trade-off parameter ρ), showcasing its ability to navigate the privacy-utility spectrum.

All experiments are performed with a ResNet18 model [HZRS16], and the client/server split is after the 6th layer. Training uses standard SGD with an initial learning rate of 0.015, momentum of 0.9, and weight decay of 10^{-4}. The number of epochs is set to $E = 10$ for CelebA and Fairface, and $E = 50$ for CIFAR-10, with a batch size $B = 128$. We find that `topk` thresholding (Section 3.3.3)), and hybrid fusion (Section 3.3.4)) perform well, and are thus used in the remaining results.

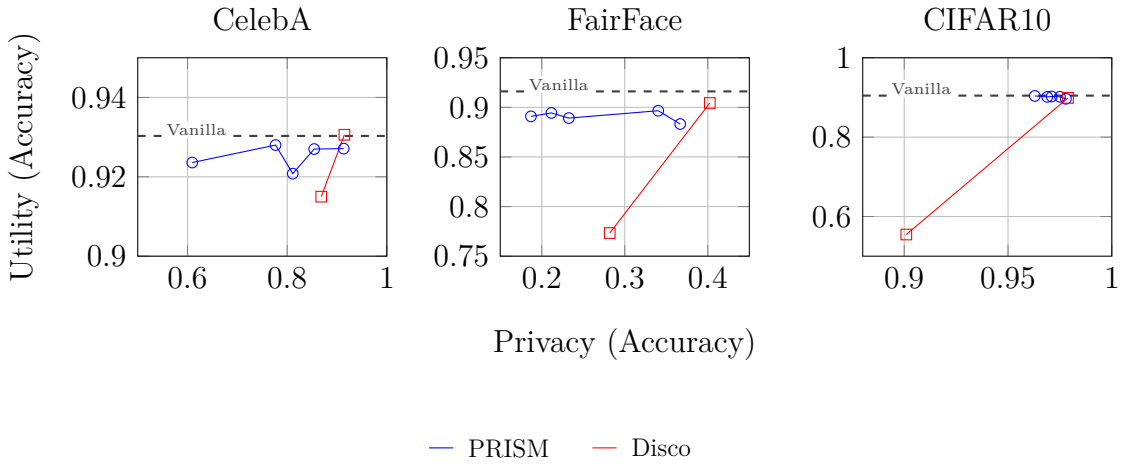

Figure 3.2: Comparison of utility vs. privacy trade-offs for PRISM, Disco, and Vanilla strategies across different datasets. Higher utility and lower privacy are generally desirable (top-left is ideal).

3.4.2 Quantitative Results Analysis

Privacy-Utility Trade-off: The Pareto plots (Figure 3.2) illustrates the privacy-utility trade-off (Task Accuracy vs. Adversary Accuracy) and core advantage of PRISM. Across all three datasets, PRISM (blue) consistently reside in a more favorable region (top-left corner) compared to DISCO, indicating that PRISM achieves an improved trade-off. For a given level of utility, PRISM provides better privacy, or conversely, for a desired level of privacy, PRISM maintains higher utility. On *CelebA* and *FairFace*, the separation is particularly evident. PRISM demonstrates configurations that significantly outperform DISCO in both metrics simultaneously or offer much stronger privacy protection with minimal utility loss. On *CIFAR-10*, while the utility levels are typically high for both methods, both methods show weak privacy metrics, which are comparable to the Vanilla SL baseline.

We posit the nature of the privacy-utility trade-off observed in Figure 3.2 is linked to the relationship between the primary task labels and potential sensitive attributes within each dataset. For instance, in *CelebA*, the task (e.g., classifying 'Smiling') and a sensitive attribute (e.g., 'Gender') are relatively independent. Both men and women can be either smiling or not smiling. This decoupling may allow privacy mech-

Table 3.2: Comparison of privacy-utility trade-offs and reconstruction quality metrics for different strategies across datasets. Utility and Privacy are classifier accuracies (higher is better).

Strategy	Utility ↑	Privacy ↓	PSNR ↓	SSIM ↓	LPIPS ↑
CelebA					
Vanilla	0.9303	0.9539	20.03	0.8742	0.4894
Disco (8)	0.9150	0.8683	11.44	0.2015	0.7352
Disco (Off)	0.9306	0.9148	13.22	0.5198	0.6860
PRISM (0.9)	0.9236	0.6094	10.60	0.1699	0.7935
PRISM (0.83)	0.9280	0.7766	10.98	0.2312	0.7557
PRISM (0.75)	0.9208	0.8112	11.90	0.3071	0.7272
PRISM (0.65)	0.9270	0.8542	11.90	0.3288	0.7130
PRISM (0.54)	0.9271	0.9134	13.70	0.5679	0.6828
FairFace					
Vanilla	0.9161	0.4188	21.18	0.8302	0.4910
Disco (8)	0.7733	0.2819	12.83	0.2598	0.7581*
Disco (Off)	0.9044	0.4024	14.79	0.6609	0.6165
PRISM (0.9)	0.8944	0.2116	12.85	0.2898	0.7472
PRISM (0.83)	0.8909	0.1870	13.12	0.3018	0.7304
PRISM (0.75)	0.8892	0.2326	13.22	0.3209	0.7389
PRISM (0.65)	0.8832	0.3667	13.72	0.5098	0.6897
PRISM (0.54)	0.8966	0.3403	14.20	0.5797	0.6698
CIFAR10					
Vanilla	0.9046	0.9190	16.54	0.8369	0.5932
Disco (8)	0.5546	0.9010	12.82	0.5075	0.6512*
Disco (Off)	0.8989	0.9789	13.70	0.6321	0.6251*
PRISM (0.9)	0.9038	0.9629	12.41	0.4762	0.6638*
PRISM (0.83)	0.9014	0.9689	12.44	0.4879	0.6520*
PRISM (0.75)	0.9024	0.9710	12.56	0.4999	0.6595*
PRISM (0.65)	0.9017	0.9750	12.54	0.5153	0.6573*
PRISM (0.54)	0.8966	0.9779	12.62	0.5244	0.6504*

anisms like PRISM to suppress gender-related features without drastically impacting the ability to classify smiles, leading to a more favorable and flexible trade-off curve. Conversely, in *CIFAR-10*, the task (e.g., identifying an 'Airplane') and a potential sensitive property (e.g., 'living') are strongly correlated or mutually exclusive (i.e. an Airplane cannot be 'living'). Features critical for identifying an airplane inherently indicate it is non-living. Attempting to obscure the 'living' property might directly interfere with recognizing the airplane, forcing a tighter and potentially less favorable trade-off where utility must be sacrificed to gain privacy. This characteristic could explain why achieving strong privacy gains at near-optimal utility appears more feasible on CelebA compared to CIFAR-10.

Detailed Metrics: Table 3.2 provides a full view of the Pareto plot observations.

- **Utility:** Vanilla SL sets the upper bound for utility as expected. DISCO (Off) achieves utility close to Vanilla, but DISCO (8) shows significant utility degradation, especially on *FairFace* and *CIFAR-10*. In contrast, PRISM consistently maintains high utility across its configurations, often very close to the Vanilla baseline, even when achieving substantial privacy gains. For instance, on *CelebA*, PRISM configurations maintain utility around 0.92-0.93 while varying adversary accuracy significantly. This highlights the effectiveness of the bypass stream (z_C) and disentangled optimization in preserving task-relevant information.

- **Privacy (Adversary Accuracy):** PRISM demonstrates the ability to significantly reduce adversary accuracy. On *CelebA*, PRISM (0.9) achieves 0.6094 adversary accuracy, far superior to Vanilla (0.9539) and DISCO (Off) (0.9148), while maintaining high utility. On *FairFace*, PRISM configurations achieve adversary accuracies as low as 0.1870, better than Vanilla (0.4188) and competitive with DISCO, but without the severe utility drop seen in Disco(8).

- **Privacy (Reconstruction Quality):** PRISM consistently yields poor reconstruction quality for the attacker, indicated by low PSNR/SSIM and high LPIPS

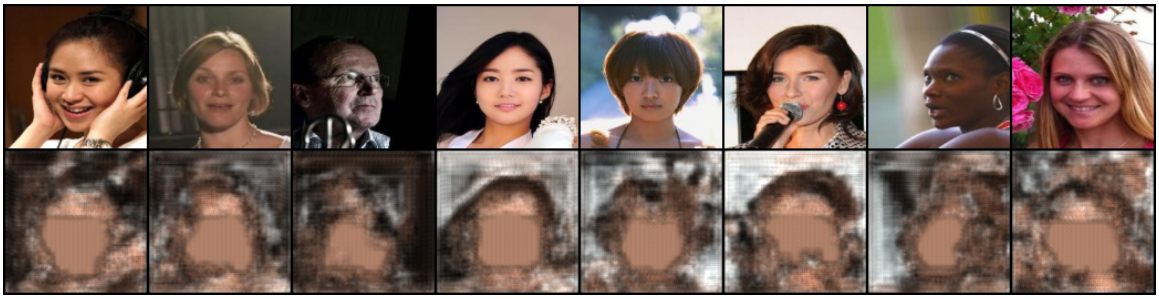

Figure 3.3: Supervised decoder reconstruction attack of PRISM on CelebA. We show the complete removal of facial features, and degradation of surrounding structure. This learned ranking of spatial and channel pruning is based on the combination of task and adversarial losses

values, often exceeding the degradation achieved by DISCO, especially at comparable utility levels. For example, on *CelebA*, PRISM (0.9) has PSNR=10.60 and SSIM=0.1699, indicating much worse reconstruction than Vanilla (PSNR=20.03, SSIM=0.8742) and also surpassing Disco (8) (PSNR=11.44, SSIM=0.2015) in this instance. This suggests that PRISM's spatio-channel masking removes structural and perceptual information from the server-bound stream z_S.

Overall, the quantitative results suggest PRISM provides a favorable and controllable privacy-utility trade-off than prior methods. It effectively mitigates both attribute inference and reconstruction attacks while largely preserving the performance on the primary task.

3.4.3 Qualitative Results Analysis

Figure 3.3 and Figure 3.4 provide visual evidence of PRISM's masking mechanism through supervised decoder reconstruction attacks on *CelebA*. In Figure 3.3, the reconstructions (bottom row) generated from the server-bound features (z_S), have task related features routed to the bypass, showing a dramatic degradation compared to the original images (top row). Crucially, salient facial features, which often carry sensitive attributes like identity or demographics, appear heavily obscured or entirely removed. Surrounding structures and backgrounds are also degraded, but the targeted removal of facial information is evident. This visually confirms adversarially

Figure 3.4: Supervised decoder reconstruction attack of PRISM on CelebA. Here, we route scores with high task loss to the server instead of bypass layers. This behavior is intended when the task itself contains little sensitive information.

guided spatio-channel masking can identify and suppress sensitive information within z_S, which aligns with the poor quantitative reconstruction metrics (low PSNR/SSIM, high LPIPS). Figure 3.4 illustrates a scenario where features associated with high task loss (potentially less sensitive but important for the main task) are routed to the server. This demonstrates the learned masking is not simply removing fixed regions but adapting based on the combined privacy and utility objective defined during training.

These qualitative results underscore PRISM's ability to perform fine-grained, content-aware masking, hindering an attacker's ability to visually reconstruct sensitive aspects of the original input from the information shared with the server.

3.4.4 Summary of Findings

The evaluation demonstrates that PRISM effectively enhances privacy in Split Learning while preserving high task utility. PRISM achieves an improved privacy-utility trade-off compared to prior works, showcasing its bypass stream, disentangled optimization, and potentially finer-grained masking. Qualitatively, it degrades the quality of reconstructed images, particularly obscuring sensitive features like faces, and validated by quantitative metrics (PSNR, SSIM, LPIPS). The bypass ratio provide control over the privacy-utility balance, allowing practitioners to adapt the level of protection based on specific needs. These results validate the architectural and methodological design choices of PRISM, positioning it as an effective mechanism for

Table 3.3: Comparison of privacy-preserving split-learning approaches. Under *Architecture*, Linear = Client→Server and U-shape = Client-in→Server→Client-out.

Feature	Vanilla [GR18]	ARL [LGY+21]	NoPeek [VSGR20]	DISCO [SCG+21]	PRISM
Architecture	Linear	Linear	Linear	Linear	U-shaped + Bypass
Privacy mechanism	–	Adv. invariance	dCor.	Channel pruning	Spatio-channel mask + route
Info sent to server	Raw z	Obfuscated z	Regularised z	Pruned z_S	Pruned z_S
Handling of sensitive info	Transmitted	Suppressed	Suppressed	Discarded	Routed z_C
Local-only info	–	–	–	–	z_C for f_C^{out}
Encoder objective	Task loss	Task + adv.	Task + reg.	Disentangled task loss	Disentangled task loss
Privacy-ratio control	No	No	No	Yes	Yes
Utility preservation	Implicit	Trade-off dense z	Trade-off reg. z	Trade-off pruned z_S	Trade-off z_S + bypass z_C

privacy-preserving collaborative learning in the Split Learning paradigm.

3.5 Discussion

The architectural and optimization design choices of PRISM lead to specific implications for privacy, performance, and their trade-offs, differentiating it from prior approaches. Table 3.3 provides a summary comparison.

3.5.1 Privacy Implications

PRISM both introduces, and adopts several strategies to mitigate information leakage to the potentially untrusted server node.

Mitigation Mechanisms

1. **Adversarially Guided Spatio-Channel Pruning of z_S:** The masking module M_θ, guided by the privacy term in $\mathcal{L}_{\text{mask}}^{\text{total}}$ (e.g., $-\mathcal{L}_{\text{adv}}$), learns to identify and suppress features in z that are most informative for the proxy adversary

f_{adv}. By setting corresponding entries in the mask M to 1 (sending to z_S) only if they are deemed necessary for utility via the server path and sufficiently non-sensitive, M_θ actively sanitizes the information flowing to the server. The spatio-channel granularity allows for targeted removal (e.g., masking a face region) rather than coarse channel removal. The resulting z_S is inherently sparser and contains less sensitive information than the original z.

2. **Selective Information Routing via Bypass Stream z_C:** This is arguably PRISM's most distinctive privacy feature. Features that are masked out ($M = 0$) are not discarded but routed into the local bypass stream z_C. This information *never traverses the network* and is never exposed to the server environment. Any sensitive information successfully isolated into z_C by the masker M_θ is thus protected from server-side snooping, limited only by the security of the client's local execution environment. This architectural separation provides a strong containment mechanism.

Information-Theoretic Perspective PRISM's operation can be interpreted through the lens of the Information Bottleneck (IB) principle [TPB00]. The goal is to create a representation (in this case, z_S) that is maximally informative about the target variable y while being minimally informative about the input x or sensitive attribute s.

PRISM effectively imposes an information bottleneck specifically on the *server communication channel*. The objective implicitly aims to:

- Minimize $I(s; z_S)$: Reduce the mutual information between the sensitive attribute s and the server-bound activation z_S. The adversarial training of M_θ directly pushes towards this goal.

- Maximize $I(y; \hat{y})$: Maximize the mutual information between the true label y and the final prediction $\hat{y}$, which depends on both z_S (via u) and z_C.

The key advantage arises from the bypass stream z_C. Conventional methods operating on a single stream z face a hard trade-off: reducing $I(s; z)$ might inadvertently

reduce $I(y; z)$ if s and y are correlated or rely on shared underlying features. PRISM circumvents this. It allows the initial representation z to potentially have high $I(s; z)$ and high $I(y; z)$ (as f_C^{in} optimizes only for task utility). The masker M_θ then attempts to partition the information:

- Route features contributing primarily to $I(s; z)$ (and less critical for $I(y; z)$ via server) into z_C.
- Route features contributing primarily to $I(y; z)$ (and less to $I(s; z)$) into z_S.
- Features contributing significantly to both might be routed to z_C (prioritizing privacy of z_S) or z_S (prioritizing utility via server), depending on the learned masking policy and the trade-off parameter ρ.

Because z_C contributes to the final prediction $\hat{y}$ via f_C^{out}, the information $I(y; z_C)$ is not lost. Thus, PRISM can aggressively reduce $I(s; z_S)$ by routing sensitive information locally, without necessarily suffering the same utility loss as methods that must globally reduce $I(s; z)$.

Furthermore, during *inference* for any specific feature element z_{ijk} where the mask $M_{ijk} = 0$, the corresponding element in z_S is zero. Conditionally, given the mask, the mutual information $I(z_{ijk}; (z_S)_{ijk} | M_{ijk} = 0) = 0$. This offers a strong, structured guarantee: *no information about the masked-out features is transmitted to the server.* This constitutes a form of perfect privacy *for those specific components*, a stronger claim than typically possible with methods relying solely on noise addition or implicit regularization.

Residual Risks and Limitations Despite its advantages, PRISM's privacy is not absolute:

- **Empirical Guarantee:** The privacy protection relies on the effectiveness of the learned mask M_θ and the representativeness of the proxy adversary f_{adv} (if used). Leakage from z_S can still occur if the mask fails to identify all sensitive features or if a real-world attacker uses different techniques than the proxy adversary.

- **Gradient Leakage:** Like most SL variants that involve gradient exchange, gradients flowing back from the server related to z_S could potentially leak information about z_S itself [GBDM20, ZH20]. PRISM's core mechanism does not directly address this channel, although techniques like gradient perturbation or compression could be applied orthogonally.

- **Client-Side Security:** The privacy of the bypassed information z_C hinges entirely on the assumption that the client's local environment (hosting Client-In and Client-Out) is secure.

- **Mask Interpretability vs. Effectiveness:** While spatio-channel masks might be more interpretable than dense embeddings (e.g., visualizing masked regions), ensuring the learned mask correctly identifies sensitive information across diverse inputs remains challenging.

3.5.2 Performance Aspects

While PRISM is designed to preserve high task utility and mask sensitive information, its structure may also improve communication costs due to increased sparsity. Although this may come at the cost of increased client computation.

Task Accuracy Several factors contribute to preserving accuracy: (1) f_C^{in} learns features guided solely by $\mathcal{L}_{\text{task}}$ using gradients from both paths, preventing direct degradation from privacy objectives. (2) z_C provides the Client-Out node f_C^{out} with unpruned features that were not filtered for privacy. This compensates for information potentially lost in z_S due to aggressive masking. (3) The fusion mechanism in f_C^{out} allows the model to leverage both the computationally intensive work done by the server on u and the detailed local information in z'_C.

We hypothesize that for a given level of privacy leakage to the server (e.g., measured by adversary success on z_S), PRISM should achieve higher task accuracy than methods like DISCO that discard masked information, because PRISM retains that information's utility locally.

Computational Overhead Compared to vanilla SL, the client must additional components. (1) The masking module M_θ. Typically, M_θ is designed to be lightweight compared to f_C^{in} and f_S. (2) Additional cost of executing the `BypassDownsample` module and the fusion operation within f_C^{out}.

The server on the other hand, processes z_S, which is sparser than z. This likely reduces the server's computational load per sample compared to vanilla SL or ARL methods processing dense z. The overall system latency depends on the relative costs and network communication.

Communication Overhead There are 4 network boundaries (2 forward, 2 gradient), and 2 local boundaries (1 forward, 1 gradient) between models.

Notably, (1) Client-In $\rightarrow$ Server: Transmits z_S. Due to masking (M contains zeros), z_S is sparse. If sparsity is exploited via sparse tensor formats or compression, the payload size can be significantly smaller than transmitting dense z in vanilla SL or ARL. The size relative to DISCO depends on the respective pruning ratios and methods. (2) Server $\rightarrow$ Client-Out: Transmits u. Size depends on the architecture of f_S. (3) Client-In $\rightarrow$ Client-Out (Local): Transmits z_C. This occurs within the client's trusted environment (potentially intra-process or inter-process communication) and has negligible network cost. (4) Gradients: Communication includes gradients for z_S (Server $\rightarrow$ Client-In) and gradients for u (Client-Out $\rightarrow$ Server). The volume of gradients related to z_S might also be reduced due to sparsity, making top-k sampling alongside gradient quantization more effective [WLY+23].

Overall network communication is likely reduced compared to vanilla SL, especially if sparsity is leveraged.

3.5.3 Privacy-Performance Trade-Off

Multiple knobs are available to tune the balance between privacy guarantees (on z_S) and overall task performance:

- **Adversarial Weight (ρ):** The parameter ρ in $\mathcal{L}_{\text{mask}}$ (Eq. (3.6)) directly controls the relative importance of utility vs. privacy when training the masker

M_θ. Lower ρ puts more weight on fooling the adversary, likely leading to more aggressive masking of z_S (better privacy on server path) but potentially relying more heavily on z_C for utility.

- **Pruning Ratio Target (r_{target}):** If using sparsity regularization in $\mathcal{L}_{aux}$ or `topk` thresholding, this target directly controls the average fraction of features routed to the bypass stream z_C. Increasing the target bypass ratio sends less information to the server while retaining more locally, which potentially improves privacy. The auxiliary loss weight (γ) can be increased to strengthen regularization penalties

- **Masking Module Architecture/Style:** The choice of `scoring_style` (channel, pixel, hybrid) and the complexity of M_θ itself influence the granularity and capability of the privacy filter.

- **Adversary Strength/Architecture:** The choice of f_{adv} implicitly defines the notion of "sensitivity" the system optimizes against.

The existence of the bypass stream z_C provides a degree of freedom in this trade-off. Unlike systems where utility must be encoded entirely within the privacy-protected stream, PRISM can maintain high utility locally via z_C even if z_S needs to be heavily sanitized for privacy.

3.5.4 Compatibility with Existing Methods

It is important to note that PRISM is not mutually exclusive with all other privacy techniques. For instance, the communication channel transmitting z_S and u could still be protected using standard encryption. Differential privacy noise could potentially be added to z_S after masking by PRISM, providing an additional layer of formal privacy guarantee, albeit likely with a further utility cost. PRISM focuses on a learning-based, structural defense at the feature level, complementing potential cryptographic or noise-based methods.

3.6 Summary

PRISM presents a novel framework for improved privacy in split learning, addressing key limitations of prior approaches. By integrating a dynamic spatio-channel masking module within a three-node U-shaped architecture featuring a local bypass stream, with a disentangled optimization strategy and explicit mask control, PRISM offers several advantages:

1. It provides fine-grained control over information sent to the server, enabling targeted suppression of sensitive features.

2. It avoids discarding potentially useful information by routing masked features locally via the bypass stream, preserving utility that would be lost in prior methods.

3. Its disentangled optimization shields the primary feature encoder from direct privacy penalties, allowing it to learn richer representations focused solely on task performance, while a specialized module handles the privacy filtering.

4. It offers flexibility through tunable parameters and the potential incorporation of different privacy objectives (adversarial or information-theoretic) and explicit mask regularization.

Theoretically, PRISM aims to create a tailored information bottleneck on the server communication channel, minimizing leakage ($I(s; z_S)$) while preserving overall task performance ($I(y; \hat{y})$) by leveraging the local path (z_C). This structured approach, particularly the zero leakage guarantee for fully masked features, offers stronger and more interpretable protection than methods relying solely on global regularization or implicit obfuscation.

While empirical results demonstrate PRISM's effectiveness in achieving better privacy-utility tradeoffs on benchmark datasets compared to existing methods, several avenues for future research remain:

1. Rigorously evaluating PRISM against a wider range of sophisticated adaptive

adversaries and reconstruction attacks, beyond the proxy adversary used for training. Standardized benchmarks [EKÇ22] could be used.

2. Developing tighter theoretical bounds on the information leakage ($I(s; z_S)$) based on the mask properties and training dynamics. Formalizing the privacy guarantees, perhaps under specific assumptions about data or adversary capabilities.

3. Exploring methods for adaptively tuning the trade-off parameters (ρ, r_{target}, γ) during training based on observed leakage or utility metrics, reducing the burden of manual hyperparameter search.

4. Further research into the performance implications of using different information-theoretic privacy objectives within $\mathcal{L}_{\text{mask}}$ and studying the impact of various fusion mechanisms (Section 3.3.4) on both utility and potential subtle privacy interactions.

5. Integrating orthogonal techniques to protect against information leakage from gradients exchanged during backpropagation.

6. Analyzing the performance of PRISM in multi-clients scenarios.

In summary, PRISM represents a novel privacy-preserving mechanism, which shows favorable privacy-utility tradeoffs in distributed learning frameworks like Split Learning. By combining feature routing and local bypassing with feature disentanglement, it offers a unique approach to mitigating the privacy-utility trade-offs in collaborative machine learning.

Chapter 4

Resource Constraints in Split Learning

4.1 Introduction

The increasing deployment of Internet of Things (IoT) devices and edge computing necessitates models capable of operating under strict resource limitations, including constraints on computation, memory, and energy. Training complex deep learning workflows directly on such devices is often infeasible. Split Learning (SL) [GR18] has emerged as a frontrunner, enabling collaborative model training across multiple data silos without centralizing raw data, thereby enhancing data privacy.

SL partitions a neural network model M at a specific 'cut layer'. The client device computes the initial layers up to the cut layer, while a server computes the remaining layers. Communication typically involves the client sending intermediate activations (the 'smashed data') to the server, and the server sending back gradients corresponding to these activations for the client to perform its backward pass and update its local model part [GR18]. This strategy inherently reduces the computational load on the client compared to FL, where the entire model is typically trained locally. Consequently, SL is often considered particularly suitable for resource-constrained settings.

However, standard SL still imposes non-negligible requirements on the client.

Clients must store intermediate activations necessary for backpropagation, compute gradients for their model portion, and maintain optimizer states (e.g., momentum buffers). These requirements can remain prohibitive for ultra-low-power devices or sensors with minimal processing and memory capabilities [LZP+24, MLJ+25]. In FL, a common approach to handle devices that are too slow or incapable of participating in a training round (stragglers) is to exclude them [LSZ+20]. This leads to a loss of their data contribution for that round and can introduce bias, especially if device capability correlates with data characteristics. This issue connects to the broader challenge of ensuring fair resource allocation in federated systems, where exclusion of certain clients can harm model equity [LSBS19]. Critically, an FL client that does not compute and send a model update provides no benefit to the global model aggregation in that round. Similarly, discarding incapable clients in standard SL wastes potentially valuable data contributions, hindering model performance and reducing the diversity of data seen by the server, which is particularly detrimental in non-IID data scenarios [ZLL+18].

To address these limitations, this chapter proposes Split Learning with Frozen Clients (SL-FC). SL-FC is a hybrid framework designed for heterogeneous environments containing clients with mixed computational capabilities:

- **Active Clients:** These clients possess sufficient resources to participate in standard SL training, performing both forward and backward passes to update their local client-side model parameters (θ_C).

- **Frozen Clients:** These represent devices with minimal resources. Their client-side model parameters are frozen, either fixed or updated infrequently (e.g., periodically receiving weights from active clients or the server). Frozen clients only perform the forward pass up to the cut layer, transmitting their computed activations (a_c) along with the corresponding data labels (y) to the server. No backward pass or model updates are performed.

- **Server:** The server manages the training process, interacting with both client types. It performs the standard SL forward and backward steps with active

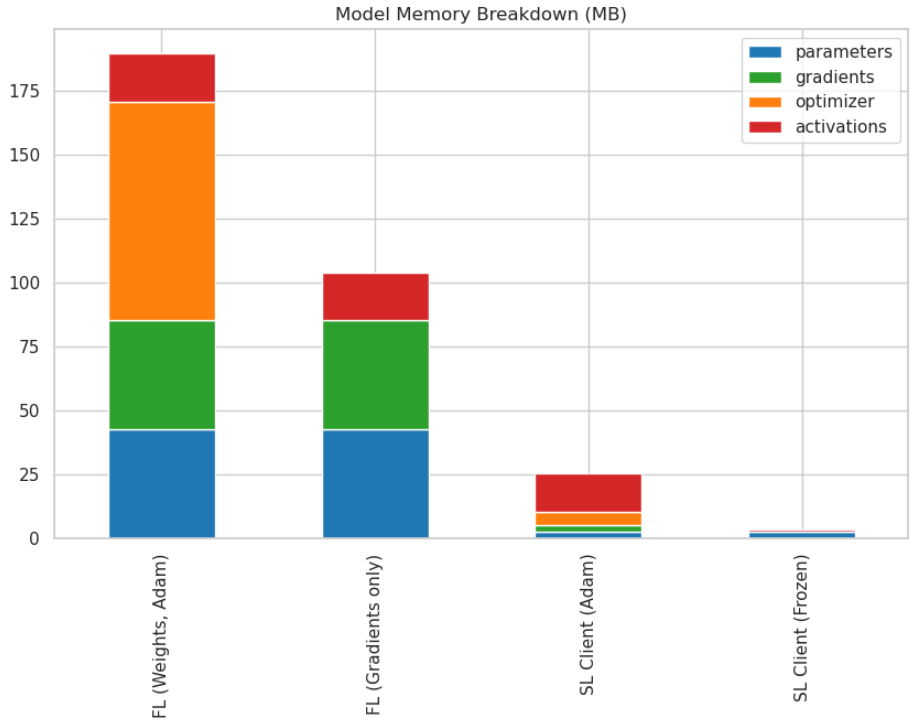

Figure 4.1: Client memory breakdown for a ResNet18 model. Since split learning allows clients to run a small portion of a model, the memory overhead is much lower. Removing optimizer states and saved activations required for backpropagation can significantly reduce overhead. Alternatively, gradient checkpointing may be used at the expense of extra computation.

clients (computing server-side outputs, loss, and sending back gradients for the cut layer activations), while only updating its own weights with frozen clients.

The core contributions of this work are:

1. A quantitative assessment of the computational and memory savings afforded to frozen clients, resulting from the elimination of gradient computation & storage, optimizer state, and potentially the need to store intermediate activations for backpropagation (illustrated conceptually in Figure 4.1).

2. Empirical validation using ResNet18 on CIFAR-10 under various data heterogeneity settings, demonstrating that SL-FC outperforms the baseline strategy of

discarding low-resource clients, preserving data diversity and improving model performance.

3. An analysis of the performance versus resource trade-offs inherent in SL-FC, highlighting its unique capability, compared to FL, to extract value from clients incapable of performing local updates during ongoing training rounds.

The effectiveness of SL-FC stems from exploiting the inherent asymmetry of the SL protocol. The server's ability to update its parameters θ_S depends on receiving client activations a_c and labels y to compute a loss. Even if a client cannot compute gradients for its own parameters θ_C, its contribution of (a_c, y) still provides a valuable learning signal for the server. This allows SL-FC to retain the data contributions from ultra-resource-constrained devices, which would be entirely lost in an FL setting or if such clients were simply excluded from standard SL.

4.2 Background

Split Learning involves splitting a model M into a client part M_C and a server part M_S at a cut layer [GR18]. The client processes input data x through M_C to produce activations a_c, which are sent to the server who processes through M_S to produce output $\hat{y}$, calculates loss $L(\hat{y}, y)$, and computes gradients $\nabla_{a_c} L$. These gradients are sent back to the client, which performs backpropagation through M_C to update its parameters θ_C. The server updates its parameters θ_S using $\nabla_{\theta_S} L$. This process reduces client computation compared to FL and enhances privacy as raw data is not shared. Vanilla SL operates sequentially, training with one client at a time, which can be slow with many clients. Parallel Split Learning (PSL) [CW22] allows multiple clients to compute their forward passes concurrently, reducing latency. Active clients in SL-FC operate similarly to PSL participants. Variations like Multi-head SL [GR18] explore architectures without client-side model synchronization. Research has also focused on improving SL's communication efficiency through techniques like sparsification, quantization [ADK+23], or binarization of activations or gradients [OLBJ25]. SL-FC

primarily targets computational and memory reduction on the client, while activation communication remains necessary.

Federated Learning [MMR+17, KMA+21] involves clients training a full model locally on their private data and periodically sending model updates (gradients or weights) to a central server for aggregation into a global model. FL does include some limitations however, such as communication overhead, computational burden on clients, and handling system and statistical heterogeneity [LSZ+20, RCZ+21]. A significant difference relevant to SL-FC is how inactive or resource-constrained clients are handled. In FL, clients that cannot complete local training and send an update within a round typically contribute nothing to the global model update for that round. In contrast, SL-FC allows frozen clients to contribute to the server model's update via their activations and labels.

SplitFed [TZEM21, TACS22] emerged as a hybrid approach aiming to combine the benefits of SL and FL. It splits the model as in SL, allowing clients to train their client-side parts in parallel, and then aggregating the client-side models using FL aggregation strategies, while the server manages the server-side model. SFL aims for lower client computation than FL and faster training than vanilla SL. However, most SFL variants still assume that participating clients perform both forward and backward passes on their client-side model and contribute updates for aggregation. SL-FC differs by introducing a class of "frozen" clients that do not perform local updates or participate in client-side model aggregation. This is not mutually exclusive to SFL, since frozen clients may contribute asynchronously to the server, although they will still be excluded from the FL aggregation.

SL-FC fits within the broader effort to enable machine learning on resource-constrained edge devices [LZP+24, MLJ+25]. Various techniques address this, including model compression, quantization, and frameworks designed for heterogeneity. One line of research specifically addresses memory and computation constraints by freezing some or all client-side model parameters. For instance, SplitFrozen [MLJ+25] partitions a large language model (LLM) such that edge devices run only forward passes on the early (frozen) layers, while a central server fine-tunes the remaining layers, dramatically cutting on-device computation and training time while improv-

ing accuracy compared to baselines that updated client weights. While the concept of SplitFrozen also explores scenarios with frozen client-side models, particularly for large language models, SL-FC introduces key distinctions relevant to broader heterogeneous learning environments. Specifically, SplitFrozen primarily considers a setup where all participating edge devices operate with frozen, pre-trained initial layers, focusing on server-side fine-tuning. In contrast, SL-FC investigates a hybrid system where a subset of clients (active clients) engage in full SL training with backpropagation, while others (frozen clients) contribute only forward-pass activations. Furthermore, our work emphasizes training from scratch to understand the dynamics of incorporating minimal-resource frozen clients on the overall learning process, rather than solely optimizing a pre-trained model. More generally, partial model training allows weak clients to train just a subset of layers that fit their capacity, freezing the rest. Lee et al. [LZP+24] propose letting each client train only as many of the last layers as it can handle, proving convergence to a neighborhood of the optimum and showing empirically that including extremely resource-limited clients yields minimal degradation compared to an ideal scenario, far outperforming exclusion. Other approaches dynamically adjust the SL cut layer based on client capabilities or use multiple fixed partition points (e.g., FedSplitX). FedLite [GKA+20] uses SL principles within FL but focuses on compressing communication. The challenge of client heterogeneity is also tackled by personalized federated learning, which aims to train models that are tailored to each client's specific data distribution, often as a way to combat client drift [FMO20]. SL-FC offers a distinct hybrid strategy by defining a minimal participation mode (forward-only) for the most constrained devices within a heterogeneous population, rather than adapting the workload for clients that can still perform backpropagation or focusing solely on communication compression.

Freezing client-side models changes the information flow: clients primarily transmit intermediate activations (and potentially labels) instead of gradients. The client's network acts as a fixed feature extractor. This reduces client communication/storage overhead (no need to receive/apply gradients) but raises privacy concerns, as an honest-but-curious server might reconstruct raw inputs from activations [GBDM20, EKÇ22]. Freezing the client-side model can inherently improve resistance to such

model inversion attacks (ResSFL [LRC+22]). However, a completely frozen extractor might degrade accuracy if not well-aligned with the task. Frameworks like ResSFL address this via privacy-aware pre-training combined with frozen-client SL, using techniques like contrastive loss or bottleneck layers to limit information leakage while maintaining accuracy. Some methods further limit exposure by having the client send loss gradients w.r.t. its output instead of labels.

An important advantage of frozen models is enabling stateless participation. Stateless clients do not maintain persistent parameters across rounds, simplifying participation for devices with sporadic connectivity or minimal memory [SSG+21], meaning they are not required to send model updates and halving communications costs. Including these clients additionally avoiding the exclusion bias towards powerful devices [LZP+24], allowing the server to train on a larger, encoded dataset. Parallel processing techniques like those in SplitFed can mitigate server-side latency [TACS22]. SL-FC leverages this by allowing frozen clients to participate statelessly, contributing data without needing local updates or state persistence.

The mechanism by which the server in SL-FC learns from frozen clients resembles concepts in Knowledge Distillation (KD) [HVD15]. The server learns a mapping from the feature representations (a_c) generated by the frozen clients' fixed models (M_{C_j}) to the correct labels (y). This is conceptually similar to learning from a fixed feature extractor or feature-based KD methods where a student learns from a teacher's intermediate features [GYMT21]. While SL-FC does not explicitly match distributions, the server learns directly from the features produced by the (fixed) client models. This perspective suggests that frozen clients provide stable, albeit potentially diverse, "teaching" signals (activations) that guide the server's learning. This contrasts with standard KD but relates to ideas like freezing model parts and using feature distillation to combat client drift in FL [LRC+22].

4.3 Methodology

We partition N clients into active clients $\mathcal{A}$ and frozen clients $\mathcal{F}$, with $|\mathcal{A}| + |\mathcal{F}| = N$. The global neural network model M is split into a client-side component M_C with

parameters θ_C and a server-side component M_S with parameters θ_S. For a client k, its specific client-side model is $M_{C_k}(x; \theta_{C_k})$, which produces activations a_{c_k} at the designated cut layer when given input x. The server model $M_S(a_c; \theta_S)$ takes these activations a_c as input and produces predictions $\hat{y}$. The objective is to minimize a loss function $L(\hat{y}, y)$ over the distributed dataset.

4.3.1 SL-FC Framework

Training proceeds in communication rounds $t = 1, \ldots, T$. In each round, a subset of active and frozen clients may participate. Active clients follow standard protocols [GR18, CW22]. Below we recap standard PSL with the introduction of frozen clients:

1. **Frozen Client $j \in \mathcal{F}$ Participation:**

   - *Optional Weight Update:* Client j may receive updated parameters $\bar{\theta}_{C_j}$ from the last active client in a round-robin scheme, or periodically from the server. These parameters remain fixed for client j during its computation in this round.
   - *Forward Pass:* Using its local data batch x_j, client j computes **only** the forward pass through its client-side model: $a_{c_{j,t}} = M_{C_j}(x_j; \bar{\theta}_{C_j})$.
   - *Communication:* Client j sends the pair $(a_{c_{j,t}}, y_j)$ containing its activations and corresponding labels to the server.
   - *Resource Usage:* Client j performs no gradient computation for θ_{C_j}, requires no storage for gradients or optimizer state (e.g., momentum, Adam parameters), and potentially reduces peak memory by not needing to store intermediate activations for a backward pass (Figure 4.1).

2. **Active Client $i \in \mathcal{A}$ Participation:**

   - *Forward Pass:* Client i computes $a_{c_{i,t}} = M_{C_i}(x_i; \theta_{C_i,t})$ for its local batch x_i using its current parameters $\theta_{C_i,t}$. It sends $a_{c_{i,t}}$ to the server.

- *Server Forward/Backward (Partial):* The server computes $\hat{y}_i = M_S(a_{c_{i,t}}; \theta_{S_t})$, the loss L_{i_t}, and the gradient of the loss with respect to the received activations, $\nabla_{a_{c_{i,t}}} L_{i_t}$. The server sends this gradient back to client i. Concurrently, the server computes $\nabla_{\theta_S} L_{i_t}$ for its own update.
- *Client Backward Pass:* Upon receiving $\nabla_{a_{c_{i,t}}} L_{i_t}$, client i performs backpropagation through M_{C_i} to compute the gradient of the loss with respect to its own parameters, $\nabla_{\theta_{C_{i,t}}} L_{i_t}$.
- *Client Update:* Client i updates its parameters using a local optimizer (e.g., SGD): $\theta_{C_i,t+1} = \theta_{C_i,t} - \eta_C \nabla_{\theta_{C_{i,t}}} L_{i_t}$.
- *Optional Weight Sharing:* Client i might share its updated weights $\theta_{C_i,t+1}$ with other clients (active or frozen) following standard protocols (e.g., round-robin).

3. **Server Update:**

- The server gathers $(a_{c_{j,t}}, y_j)$ from participating frozen clients $j \in \mathcal{F}$ and implicitly holds the necessary information $(a_{c_{i,t}}, y_i)$ from participating active clients $i \in \mathcal{A}$ (as it computed L_{i_t}).
- For each frozen client j's data, the server computes the forward pass $\hat{y}_j = M_S(a_{c_{j,t}}; \theta_{S_t})$, loss L_{j_t}, and the gradient $\nabla_{\theta_S} L_{j_t}$.
- For each active client i's data, the server uses the gradient $\nabla_{\theta_S} L_{i_t}$ computed during step 2 of active client participation.
- The server aggregates the gradients for its parameters by averaging:

$$\nabla_{\theta_{S_t}} = \frac{1}{|\mathcal{A}_{part}| + |\mathcal{F}_{part}|} \left(\sum_{i \in \mathcal{A}_{part}} \nabla_{\theta_S} L_{i_t} + \sum_{j \in \mathcal{F}_{part}} \nabla_{\theta_S} L_{j_t} \right) \tag{4.1}$$

  where $\mathcal{A}_{part}$ and $\mathcal{F}_{part}$ are the sets of participating clients in round t.
- The server updates its parameters: $\theta_{S_{t+1}} = \theta_{S_t} - \eta_S \nabla_{\theta_{S_t}}$.

4.3.2 Computational and Memory Benefits for Frozen Clients

The SL-FC procedure implicitly optimizes a global objective function over the server parameters θ_S and the parameters of the active clients $\{\theta_{C_i}\}_{i\in\mathcal{A}}$. The contribution of frozen clients $j \in \mathcal{F}$ influences the optimization of θ_S through their fixed client-side parameters $\bar{\theta}_{C_j}$:

$$\min_{\theta_S,\{\theta_{C_i}\}_{i\in\mathcal{A}}} \mathbb{E}_{k\sim\mathcal{D}_{total}}[L(M_S(M_{C_k}(x_k;\theta_{C_k});\theta_S), y_k)] \tag{4.2}$$

where $\mathcal{D}_{total}$ represents the union of data across all clients, and $\theta_{C_k} = \bar{\theta}_{C_k}$ (fixed parameters for the round) if $k \in \mathcal{F}$. While the server learns from the combined data distribution, the parameters θ_C are only updated for the active subset $\mathcal{A}$.

The primary advantage for frozen clients is the reduction in resource requirements: (1) No Gradient Computation. Eliminates the backward pass through M_{C_j}. (2) No Gradient Storage. Avoids storing gradients for θ_{C_j}. (3) No Optimizer State. Frees memory otherwise used for optimizer variables (e.g., momentum buffers, Adam/Yogi accumulators). (4) Reduced Activation Storage. Peak memory usage can be significantly lowered as intermediate activations within M_{C_j} may only need to persist for the duration of the forward pass, potentially avoiding the need to store them for backpropagation (depending on the implementation details).

SL-FC extracts learning signals from non-updating clients, a capability absent in FL where inactive clients contribute nothing to a given training round. Compared to standard SL, it lowers the barrier to participation, enabling devices incapable of performing backpropagation to contribute valuable data. The server learns to interpret potentially diverse feature representations generated by these fixed client models, effectively using their data to improve θ_S. This process can be viewed as the server adapting to multiple, fixed feature extractors, guided by the labels provided by the frozen clients. Under non-IID conditions where active client models might drift, the stable feature representations from frozen clients might act as a regularizer for the server model, promoting better generalization than if only the potentially skewed active client data were used [LRC+22].

4.4 Evaluation

This section presents an empirical evaluation of the proposed Split Learning with Frozen Clients (SL-FC) framework. The primary goal is to assess the impact of including forward-only frozen clients on overall model performance compared to standard Split Learning and the baseline scenario of excluding resource-constrained clients. We investigate this under varying degrees of data heterogeneity.

4.4.1 Experimental Setup

- **Model and Dataset:** A ResNet18 architecture [HZRS16], trained on CIFAR-10 [KH+09] is used. The model was split at the output of the 6th layer as the client-side model M_C, and the remaining layers as the server-side model M_S. We note that while converting Batch Normalization layers to Group Normalization [HPMG20] produce better scores, we keep Batch Normalization for these experiments.

- **Data Partitioning and Heterogeneity Simulation:** The CIFAR-10 training dataset was distributed among $N = 10$ clients. To simulate real-world conditions, non-IID partitions are used [HQB19, RCZ+21]. Heterogeneity was induced using two methods:

    1. **Dirichlet Distribution:** Client label distributions were generated using a Dirichlet distribution with concentration parameters $\alpha \in \{1.0, 0.5, 0.3, 0.1\}$. Lower values of α correspond to higher degrees of non-IID data skew [LSWX22].
    2. **Extreme Label Skew:** An extreme case was simulated where each client exclusively held data corresponding to only 2 specific classes (Num Labels = 2).

To evaluate SL-FC, three distinct client configurations were compared: **All Active:** Represents standard Split Learning where all 10 clients are active and perform

both forward and backward passes. This serves as an upper-bound performance reference. **SL-FC (50%):** Represents the proposed SL-FC framework, where 5 clients are active and 5 are frozen (forward-pass only). **Discarded:** Represents the baseline scenario where the 5 resource-constrained clients are simply excluded from training. This simulates the common practice when devices cannot meet the requirements of standard SL or FL.

All experiments were run for 50 epochs. Optimization was performed using Stochastic Gradient Descent (SGD) with nesterov momentum set to 0.85. The initial learning rate was set to 0.015, with weight decay applied. A Cosine Annealing learning rate scheduler adjusted the LR dynamically, reaching a minimum of 1×10^{-5}. The impact of client drift was studied by varying the number of local update steps performed by active clients in each communication round (K or 'Steps' $\in \{1, 5, 25\}$). The primary evaluation metric was the final test accuracy achieved on the CIFAR-10 test set after 50 epochs.

A simple round-robin mechanism was implemented for managing client-side parameters. In the SL-FC (10/5) setup, frozen clients inherited and used the client-side model parameters from the last active client that participated in the sequence for their forward pass computation in a given round.

4.4.2 Results Analysis

The empirical results comparing the performance of SL-FC (10/5) against the fully active (10/0) and client-discarding (5/0) baselines under varying heterogeneity conditions are presented below.

Performance under Dirichlet Heterogeneity: Table 4.1 summarizes the test accuracies achieved under Dirichlet-induced non-IID conditions for $K = 1$ local step.

- **Benefit of SL-FC vs. Discarding Clients:** A key finding is the substantial advantage of SL-FC (10/5) over the client-discarding baseline (5/0) across moderate levels of heterogeneity. For $\alpha = 1.0$ (mild non-IID), $\alpha = 0.5$, and $\alpha = 0.3$, the SL-FC configuration achieves absolute accuracy gains of +32.33%,

Table 4.1: Test accuracy (%) on CIFAR-10 with ResNet18 (split layer 6, steps=1, Dirichlet partition)

α	**All Active**	**SL-FC (50%)**	**Discarded**	**Gain**
1.0	92.26	87.06	58.94	+28.12
0.5	90.24	85.01	52.64	+32.37
0.3	90.14	81.92	50.56	+31.36
0.1	70.68	51.84	31.47	+20.37

Table 4.2: Test accuracy (%) on CIFAR-10 with ResNet18 (split layer 6, num labels = 2)

Steps	**All Active**	**SL-FC (50%)**	**Discarded**	**Gain**
25	10.00	10.00	19.49	-9.49
5	29.82	21.47	23.46	-1.99
1	35.28	22.07	20.98	+1.09

+30.16%, +28.01%, and +20.37%, respectively, compared to the 5/0 baseline. This strongly validates the hypothesis that incorporating frozen clients provides significant value by allowing their data to contribute to the server-side model training, mitigating the negative impact of data loss from exclusion.

- **Impact of Heterogeneity Level (α):** Model performance generally degrades for all methods as the degree of heterogeneity increases (i.e., as α decreases). However, the significant relative advantage of SL-FC (10/5) over the 5/0 baseline persists robustly for all $\alpha \geq 0.1$, suggesting that the benefit of retaining data diversity via frozen clients is particularly crucial in non-IID settings.

- **Impact of Local Steps (K):** While Table 4.1 shows results for $K = 1$, experiments conducted with $K = 5$ and $K = 25$ (detailed results found in Appendix A.1) showed similar relative performance patterns under Dirichlet partitioning for $\alpha \geq 0.1$. Specifically, the 10/5 configuration consistently and significantly outperformed the 5/0 baseline, although absolute accuracies varied with K, generally decreasing as K increased due to client drift.

Performance under Extreme Label Heterogeneity (Num Labels = 2): Table 4.2 presents the results for the extreme non-IID scenario where each client possessed data from only two distinct classes.

- **Overall Difficulty:** This setting proved highly challenging, resulting in considerably lower test accuracies across all configurations compared to the Dirichlet partitioning.
- **Consistency of SL-FC Benefit:** In this extreme case, the benefit of SL-FC (10/5) over discarding clients (5/0) was less consistent and depended on the number of local steps K. For $K = 1$, SL-FC provided a marginal gain (+1.09%). However, for $K = 5$ and $K = 25$, SL-FC performed slightly worse than the 5/0 baseline (-1.55% and -2.79% respectively). This suggests that when faced with extreme label polarization combined with potentially significant client model divergence (especially at higher K), the server model may struggle to effectively leverage the fixed feature representations provided by the frozen clients. The fully active (10/0) configuration, while still performing poorly overall, generally maintained the highest accuracy.

Hypothesized Dynamics in Extreme Heterogeneity Under extreme heterogeneity $\alpha = 0.1$, or Num Labels = 2, we observe diminished or negative performance benefits compared to the Discarded baseline, particularly with an increased number of local steps (K). This may be attributable to interplay between client specialization, and server-side adaptation.

In the **Discarded scenario**, each active client is exposed exclusively to data from only two distinct labels. This specialization likely encourages their client-side models (M_C) to optimize into feature extractors tailored for their two-label classification tasks. The server model (M_S) subsequently learns to map these specialized features to the correct outputs. The overall system may not achieve generalization across all classes, however it may attain reasonable performance over the union of labels actively trained by the active clients. The accuracy on the test set then might mask a relatively robust performance on these “seen” labels.

Conversely, in the **SL-FC scenario**, introducing frozen clients, each possessing data from only two labels with a fixed client-side model (M_{C_j} and parameters $\bar{\theta}_{C_j}$), presents challenges for the server:

1. **Feature Misalignment:** Features a_{c_j} generated by frozen clients (M_{C_j}) are optimized for specific labels and do not adapt during training. If these labels (and optimal feature space) differ significantly from active clients, or the broader distribution the server is implicitly trying to model, these fixed activations may offer little useful information. This could even be misleading for the server while learning other labels.

2. **Conflicting Signals:** The server updates its parameters (θ_S) based on gradients derived from both active clients (θ_{C_i}) which co-adapt with θ_S, and frozen clients where θ_S must adapt to static, pre-computed features a_{c_j} from $M_{C_j}(x_j; \bar{\theta}_{C_j})$. If the fixed features from frozen clients are suboptimal, they may act as noise and the server might struggle to reconcile these disparate signals. This could hinder its ability to learn effectively which dilutes the quality of the overall learning signal.

In essence, under extreme label skew, potential benefits of data diversity may be overshadowed by the cost of integrating potentially misaligned or unhelpful fixed representations. This is particularly pronounced when active clients are prone to divergence due to training on polarized data.

4.4.3 Summary of Evaluation

We demonstrate SL-FC is a highly effective strategy for improving collaborative learning performance in environments characterized by client resource heterogeneity and moderate label skew ($\alpha \geq 0.1$ in Dirichlet simulation). By enabling participation from ultra-resource-constrained devices via a forward-only mechanism, SL-FC outperforms the common practice of client exclusion, validating the principle that leveraging diverse data, even from non-updating clients, is beneficial. While SL-FC naturally incurs a performance penalty compared to an ideal system with fully capable clients,

it offers a compelling resource-accuracy trade-off. The benefits appear less consistent under extreme label skew conditions, particularly when combined with many local update steps, suggesting potential limitations related to server adaptation and client model divergence. Overall, the results strongly support SL-FC as a practical approach to be included in building more robust distributed learning systems.

4.5 Discussion

Resource vs. Accuracy Trade-off: SL-FC presents a notable operating point for heterogeneous systems. Frozen clients achieve significant computational and memory savings (Figure 4.1), enabling their participation. The system trades a moderate accuracy reduction (compared to 10/0) for this inclusivity, and can achieve better performance than client exclusion (5/0) in certain heterogeneous settings. This observation aligns with findings in related literature suggesting that incorporating constrained clients, even with limitations like frozen layers or partial training, is often preferable to their exclusion [LZP+24, MLJ+25].

Communication Overhead: Both frozen and active clients transmit activations a_c and labels y per batch, while an active client receives back gradients $\nabla_{a_c} L$. The relative communication cost depends on the dimensionality of activations and labels versus gradients. The elimination of the downlink gradient transmission ($\nabla_{a_c} L$) could save up to 2× bandwidth for frozen clients. Furthermore, the total data transmitted by frozen clients can be less than typical FL model updates. Therefore, while not the primary focus, SL-FC can maintain reasonable communication efficiency, with its main benefit lying in computational and memory savings on the client side.

Implications and Use Cases: SL-FC is suited for heterogeneous IoT and edge computing scenarios characterized by a mix of training capable devices and simple sensors, especially under moderate data heterogeneity. This is not mutually exclusive to alternatives like SplitFed [TACS22], providing more diverse data for the server, although frozen clients are still excluded from client model aggregation. Moreover,

SL-FC facilitates **stateless participation** for frozen clients, which can simplify the integration of devices with minimal memory or sporadic connectivity, similar in concept to Federated Reconstruction [SSG+21]. The server learns from the feature extractors provided by frozen clients, which might act as a form of regularization against the drift observed in active clients models, especially in non-IID settings [LRC+22].

Privacy Considerations: While SL inherently avoids sharing raw data, the transmission of activations (a_c) from frozen clients directly to the server introduces potential privacy risks, as these activations might be vulnerable to inference or reconstruction attacks [GBDM20, EKÇ22, LRC+22]. Although SL-FC does not explicitly incorporate privacy-enhancing technologies, techniques developed for SL privacy, such as adding bottleneck layers [VSGR20], feature pruning/masking (Chapter 3) [SCG+21], or privacy-aware pre-training of client models (as in ResSFL [LRC+22]), these could be integrated into the SL-FC framework to mitigate leakage risks. The unidirectional information flow (client $\rightarrow$ server activations/labels) simplifies the protocol compared to bidirectional gradient, possibly beneficial for the server, preventing gradient leakage to clients.

4.6 Summary

This chapter introduced Split Learning with Frozen Clients (SL-FC), a novel hybrid framework designed to address the challenge of incorporating ultra-resource-constrained devices into collaborative learning environments. By defining a distinct 'frozen' client role characterized by forward-pass-only computation and transmission of activations and labels, SL-FC significantly reduces the computational and memory requirements for these devices. This enables their participation in training, allowing the system to leverage their data contributions through server-side model updates.

Our empirical evaluation using ResNet18 on CIFAR-10 under various non-IID data distributions demonstrated the efficacy of SL-FC. In settings with moderate data heterogeneity, SL-FC substantially outperformed the baseline strategy of discarding resource-constrained clients, yielding significant improvements in test accuracy. This

confirms that retaining data diversity, even from non-updating clients, is beneficial for model performance. While SL-FC naturally exhibits a performance gap compared to a system with fully active clients, it presents a practical resource-accuracy trade-off. This exploits the structure of Split Learning to extract learning signals from clients incapable of local updates, a capability not present in standard Federated Learning.

Future research directions may include: adaptive client roles, transitioning clients between active and frozen states based on real-time resource availability, network conditions, or estimated data value. Determining the optimal split point in SL-FC settings, or possibly dynamic split points for differing client capabilities. Privacy integrations (e.g., differential privacy, or activation pruning), for analyzing potential security threats like poisoning attacks targeting frozen client activations.

In summary, SL-FC offers a pragmatic solution for handling resource heterogeneity. By enabling inclusive participation, it provides more robust, and data-efficient collaborative learning systems in real-world edge computing environments.

Chapter 5

Federated Optimization

5.1 Introduction

Federated learning considers the problem of optimizing a global model across many distributed clients, each with its own local data. Formally, we minimize a composite objective $F(x) = \frac{1}{m}\sum_{i=1}^{m} F_i(x)$, where $F_i(x)$ is the loss on client i's data. A classic approach, **FedAvg** (Federated Averaging), allows each client to perform multiple local SGD updates on F_i before averaging models on the server [RCZ+21]. While FedAvg is communication-efficient, it suffers in heterogeneous settings due to *client drift* (local models diverging from the global optimum) and lack of adaptivity [WCX+21]. In particular, FedAvg can struggle with heavy-tailed gradient noise and non-IID data, motivating methods that adapt learning rates using accumulated gradient information [HQB19, RCZ+21].

Recent works propose the **FedOpt** framework, where the server applies an optimizer (like momentum or adaptive methods) to the aggregated updates [RCZ+21]. Notably, *FedAvgM* adds server-side momentum [HQB19] to dampen oscillations, and *FedAdam* and *FedYogi* apply Adam/Yogi-style adaptive updates on the server. These methods often show improved stability and convergence in heterogeneous networks. However, existing adaptive federated optimizers (FedAdam/Yogi) operate on each model coordinate independently, ignoring the layer structure of neural networks.

In this section, we apply Muon (MomentUm Orthogonalized by Newton-Schulz)

[JJB+24] with the aim to provide evidence that it will work as an outer-loop optimizer. Muon is a recent optimizer advancement which leverages the **matrix structure** of neural network layers. Muon applies standard momentum on aggregated updates, then performs a zero-power orthogonalization of each weight matrix update via a Newton-Schulz (NS) iteration [JJB+24]. Intuitively, Muon replaces the raw momentum update G with the nearest semi-orthogonal matrix O (with $O^TO = I$ or $OO^T = I$) in Frobenius norm. This procedure normalizes the update's spectral properties, making all singular values equal (raising them to the 0-th power). By doing so, Muon adjusts the effective step size per direction, potentially improving convergence in ill-conditioned settings while preserving the update's direction. Empirically, Muon has achieved faster training than AdamW on vision and language tasks, as shown on CIFAR-10 and nanoGPT speedruns [JJB+24].

The remainder of this chapter provides a formal introduction of Muon in federated optimization, reviewing federated optimization and baseline methods (FedAvgM, FedAdam, FedYogi, etc.), defining the Muon optimizer and its algorithmic steps, and providing convergence guarantees for federated learning with Muon (server) and SGDM (clients) under standard assumptions. Finally, we analyze the role of the zero-power orthogonalization operator, showing how it acts as a form of low-rank orthogonal projection and discussing its effect on optimization and generalization, with comparisons to prior methods. Empirically, we show Muon works as a federated optimizer in settings with highly non-IID data distributions.

5.2 Background

We consider m clients each with loss $F_i(x) = \mathbb{E}_{\zeta\sim\mathcal{D}_i}[f_i(x;\zeta)]$, where $\mathcal{D}_i$ is client i's data distribution. The goal is to minimize $F(x) = \frac{1}{m}\sum_{i=1}^{m} F_i(x)$. Due to privacy or communication constraints, the server orchestrates training in rounds. In each round t, a subset S_t of clients is sampled (we focus on full participation $S_t = [m]$ for simplicity of analysis). The server sends the current model x_t to clients in S_t. Each client $i \in S_t$ initializes $x_{t,i}^{(0)} = x_t$ and performs K steps of local optimization (SGD

or another client optimizer) on F_i, yielding $x_{t,i}^{(K)}$. The updates are sent to the server, which aggregates them (e.g., by weighted averaging) to obtain a new global model x_{t+1} [RCZ+21].

In FedAvg [MMR+17], the server simply takes a weighted average of the final client models: $x_{t+1} = \sum_{i \in S_t} \frac{n_i}{n_S} x_{t,i}^{(K)}$, where n_i is client i's data size and $n_S = \sum_{i \in S_t} n_i$. This can be written as $x_{t+1} = x_t - \Delta_t$, where $\Delta_t = x_t - \sum_i \frac{n_i}{n_S} x_{t,i}^{(K)}$ is the average model update (the negative of a pseudo-gradient). FedAvg is equivalent to one step of gradient descent on $F(x)$ using the aggregated update (with an *implicit* server learning rate of 1). FedAvgM extends this by applying momentum at the server [HQB19]. Let v_t be the server's momentum buffer (initialized $v_0 = 0$). After computing Δ_t, FedAvgM updates:

$$v_t = \beta\, v_{t-1} + \Delta_t, \tag{5.1}$$

$$x_{t+1} = x_t - \eta\, v_t, \tag{5.2}$$

where $\beta \in [0, 1)$ is the momentum coefficient and η is the server learning rate. (FedAvg is recovered by $\beta = 0$ and $\eta = 1$.) This *heavy-ball* momentum accumulates past updates to smooth out oscillations, and was empirically shown to improve convergence on heterogeneous data [HQB19, RCZ+21, WCX+21].

FedAvgM is a special case of a broader **FedOpt** framework that decouples the *client optimizer* (used for local updates) and the *server optimizer* (used to update the global model) [RCZ+21]. In FedOpt, the server treats Δ_t as a gradient and applies any optimizer to update x_t. This is applied with an adaptive optimizer on the server, analogous to Adam, Yogi, or AdaGrad (called FedAdam, FedYogi, FedAdagrad). For example, **FedAdam** maintains first and second moment estimates m_t and v_t (as in Adam) of the aggregated pseudo-gradients $g_t = -\Delta_t$, and updates the global model with $m_t/(\sqrt{v_t} + \epsilon)$. **FedYogi** is similar to FedAdam but uses a different second-moment update (Yogi) to control growth of v_t. These adaptive methods can achieve better accuracy and stability than FedAvg/M in practice, especially for NLP tasks with heavy-tail gradients [RCZ+21]. Convergence analyses in nonconvex settings show FedAdam/Yogi attain the same $O(1/\sqrt{T})$ rate as centralized Adam under standard assumptions, while handling client drift with appropriate learning rate choices.

Techniques like SCAFFOLD [KKM+20] introduce control variates to correct client drift, FedProx [LSZ+20] and FedExProx [LAR24] adds proximal terms to limit local model deviation. These are orthogonal to our focus and can potentially be combined with Muon. We also note DiLoCo (Distributed Low-Communication) [DFR+23, DDR+25] training, a recent framework that divides training into "inner" local epochs and "outer" global updates on separate clusters. DiLoCo is effectively FedAvgM with large K and uses Nesterov momentum as the server optimizer, which was found to yield the best convergence in their setting. The Muon optimizer could serve as a drop-in replacement for the server optimizer (OuterOpt) in such frameworks, offering a new way to incorporate *layer-wise adaptive updates* in federated training. While we focus on non-IID federated use-cases, much of the following discussion applies to distributed training as well.

5.3 FedMuon - The Muon Optimizer

Muon is an optimizer for 2D weight parameters (e.g. weight matrices of dense/linear layers). It treats other parameters (bias vectors, embedding matrices, normalization parameters, etc.) with a standard method (SGD with momentum or AdamW as in the original implementation) [JJB+24]. We focus on the **server-side update** in federated training: after clients' updates are aggregated, the server applies Muon to update the global model. While Muon has been used primarily in centralized training, our interest is in its federated application as the server optimizer.

Overview: At round t, let W_t be a weight matrix in the global model (e.g., the hidden layer weights of shape $n \times m$). The server has an accumulated momentum M_{t-1} for W (initialized $M_0 = 0$). After receiving client updates, it computes the average update Δ_t for W. Muon then performs:

1. **Momentum step:** $M_t = \beta\, M_{t-1} + \Delta_t$. Here β is the server momentum. This is identical to FedAvgM's momentum accumulation [HQB19].

2. **Orthonormalization (zero-power) step:** Compute $U_t = \mathrm{Ortho}(M_t)$, the

Table 5.1: Overview of federated optimizers. $g_t = -\Delta_t$ denotes the server's effective gradient (negative aggregated update). Muon performs a post-processing $\mathrm{Ortho}(v_t)$ on momentum updates for weight matrices.

Optimizer	Server update	Adaptivity	Notes
FedAvg [RCZ+21]	$x_{t+1} = \sum_i \frac{n_i}{n_S} x_{t,i}^{(K)}$	-	No server learning rate ($\eta = 1$ equivalent).
FedAvgM [HQB19]	$v_t = \beta v_{t-1} + \Delta_t$	momentum	$\beta \approx 0.9$; reduces drift.
	$x_{t+1} = x_t - \eta v_t$	(no second moment)	
FedAdam [RCZ+21]	$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$	adaptive	Uses per-coordinate first/second
	$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$		moment accumulators.
	$x_{t+1} = x_t - \eta \frac{m_t}{\sqrt{v_t}+\epsilon}$		
FedYogi [RCZ+21]	$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$	adaptive	Controls v_t growth (for heavy-tail gradients).
	$v_t = v_{t-1} - (1-\beta_2)\,\mathrm{sign}(v_{t-1} - g_t^2)\, g_t^2$		
	$x_{t+1} = x_t - \eta \frac{m_t}{\sqrt{v_t}+\epsilon}$		
DiLoCo [DFR+23]	$y_t = \beta y_{t-1} + g_t$	Nesterov momentum	Large K
	$x_{t+1} = x_t - \eta(\beta y_{t-1} + g_t)$		reduces comm. by 500×.
FedMuon	$v_t = \beta v_{t-1} + \Delta_t$; $U_t = \mathrm{Ortho}(v_t)$	adaptive (matrix-wise)	Uses **orthonormalized** update U_t for 2D layers
	$x_{t+1} = x_t - \eta\, U_t$		(momentum for others).

orthogonalized update matrix. By definition, U_t is the matrix closest to M_t (in Frobenius norm) such that either $U_t^T U_t = I$ or $U_t U_t^T = I$ [JJB+24]. In other words, U_t has the same dimensions as M_t and has *orthonormal columns* (if W is "tall", $n \geq m$) or *orthonormal rows* (if W is "wide", $n < m$). This operation sets all singular values of M_t to 1, effectively raising its singular values to the 0th power. We discuss $\text{Ortho}(M_t)$ in more detail below.

3. **Gradient descent step:** Update the weight matrix: $W_{t+1} = W_t - \eta\, U_t$, where η is the server learning rate. This uses the orthonormalized momentum U_t in place of the raw update.

For parameters that are not 2D matrices (e.g., bias vectors or embeddings), we skip step 2 and use the standard momentum update: $W_{t+1} = W_t - \eta\, M_t$ (or Adam-style update in practice for embeddings [JJB+24]). Alternatively, if the number of dimensions is > 2 (e.g. Conv layers), step 2 may be applied to a flattened representation. Crucially, these parameters use the same momentum coefficient β as the matrices. This approach recognizes that the benefits of Muon's orthogonalization are most pronounced for weight matrices connecting dense layers [Ber25], where preserving orthogonal directions is hypothesized to improve training dynamics.

Definition of the Ortho operator: The core of Muon is the operator $\text{Ortho}(G)$ that projects a matrix G onto the manifold of semi-orthogonal matrices. Formally, for a given matrix $G \in \mathbb{R}^{n \times m}$, we define:

$$\text{Ortho}(G) \;=\; \arg\min_{O} \{ \|O - G\|_F : \text{ either } O^T O = I_m \text{ or } O O^T = I_n \}, \tag{5.3}$$

choosing $O^T O = I_m$ when $n \geq m$ (orthonormal columns) and $O O^T = I_n$ when $n < m$ (orthonormal rows) [JJB+24]. It can be shown that if $G = U \Sigma V^T$ is the singular value decomposition (SVD) of G (with $U \in \mathbb{R}^{n \times r}$, $V \in \mathbb{R}^{m \times r}$, $r = \text{rank}(G)$, and $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_r)$), then:

$$\text{Ortho}(G) \;=\; U I_r V^T, \tag{5.4}$$

i.e. we replace Σ by the identity matrix I_r (for the top r singular values) [JJB+24]. We adopt the standard definition where the positive singular values are mapped to 1,

ensuring $\text{Ortho}(G) = UI_rV^T$ is uniquely defined even if $r < \min(n, m)$. Intuitively, $\text{Ortho}(G)$ has the same singular vectors as G but **all singular values set to 1**. This is the closest matrix to G on the Stiefel manifold of semi-orthogonal matrices. In particular, if G is full-column rank (when $n \geq m$), $\text{Ortho}(G) = G(G^TG)^{-1/2}$, if G is full-row rank ($n < m$), $\text{Ortho}(G) = (GG^T)^{-1/2}G$. These forms come from solving the normal equations for O with orthonormality constraints (they also coincide with the limit of steepest descent on G under spectral norm regularization, as discussed later).

Efficient approximation via Newton-Schulz iteration: Computing an SVD for every update would be computationally expensive. Muon instead uses a fixed **Newton-Schulz (NS) iteration** to approximate $\text{Ortho}(G)$ efficiently [JJB+24]. The Newton-Schulz method is a well-known iterative technique for computing matrix inverses or inverse square-roots [BB71, Hig08]. In our context, we want to compute $G(G^TG)^{-1/2}$ (or $(GG^T)^{-1/2}G$). The NS iterations provide a stable way to approximate $A^{-1/2}$ for $A = G^TG$ or $A = GG^T$ without explicit eigen-decomposition.

In practice, Muon's implementation uses a 5-step polynomial iteration denoted `NewtonSchulz5` [JJB+24]. We initialize $X_0 = G/\|G\|_F$. Then for $t = 0, 1, \ldots, 4$:

$$X_{t+1} = a\, X_t + (b\, A_t + c\, A_t^2)\, X_t, \tag{5.5}$$

where $A_t = X_t^T X_t$ (if $n \geq m$, applied to G^T) or $A_t = X_t X_t^T$ (if $n < m$, applied to G), and with fixed coefficients (a, b, c) tuned for convergence. In the released implementation, $a = 3.4445$, $b = -4.7750$, $c = 2.0315$ [JJB+24] chosen (via analysis and tuning) to accelerate convergence of the NS iteration in ~ 5 steps. If $n > m$ we apply the iteration to $X = G^T$ (so $A = X_t^T X_t$) and then transpose the result at the end, if $n \leq m$, we apply it directly to G ($A = X_t X_t^T$). After 5 iterations, $X_5 \approx G(G^TG)^{-1/2}$ (or $(GG^T)^{-1/2}G$). In practice, this procedure achieves an accurate orthonormalization (the Frobenius norm error $\|X_5^T X_5 - I\|_F$ or $\|X_5 X_5^T - I\|_F$ is small) and is stable in bfloat16 precision. We note that NS is used only to compute the *direction* $U_t = \text{Ortho}(M_t)$. It normalizes the update magnitude and does not require computing any eigenvalues directly. After obtaining U_t, Muon updates $W_{t+1} = W_t - \eta\, U_t$. This completes one optimizer step (or federated round) for that weight

matrix. The same procedure is applied to each trainable 2D weight matrix in the model. Pseudocode for the server update (for one round and one parameter) is given in Algorithm 1.

Algorithm 1 Muon Server Update for a Weight Matrix W (One Federated Round)

Require: $W_t \in \mathbb{R}^{n \times m}$, M_{t-1}, Δ_t, η, β

Ensure: W_{t+1}, M_t

$M_t \leftarrow \beta \cdot M_{t-1} + \Delta_t$ ▷ Momentum accumulation

if $\dim(W_t) >= 2$ **then**

    $U_t \leftarrow \texttt{NewtonSchulz5}(M_t)$ ▷ Approximate orthonormalization [JJB+24]

else

    $U_t \leftarrow M_t$ ▷ Skip orthonormalization for vector/scalar

end if

$W_{t+1} \leftarrow W_t - \eta \cdot U_t$ ▷ Model update

Remark: The orthonormalization step can be seen as applying an adaptive preconditioner to the gradient. Unlike Adam/Yogi which use a diagonal preconditioner (scaling each weight individually based on past squared gradients), Muon uses a **matrix preconditioner** $(G^TG)^{-1/2}$ or $(GG^T)^{-1/2}$ that *couples* the weights. This coupling respects the linear layer structure, potentially capturing interactions between weights. Muon's update U_t is invariant to uniform scaling of M_t (scaling M_t by any $\alpha > 0$ yields the same $\text{Ortho}(M_t)$), which means the step size is effectively determined by η and is not sensitive to the overall norm of the gradient, only its direction matters. This property can mitigate issues of learning rate tuning and gradient magnitude variability across layers, which is desirable in settings where tuning learning rates is prohibitively expensive, due to computational, or privacy constraints [LSZ+20, KMA+21].

5.4 Convergence Analysis

We now analyze federated learning with Muon as the server optimizer and SGD with momentum (SGDM) as the client optimizer. We aim to show convergence under standard assumptions for nonconvex objectives, following the spirit of Reddi *et al.* [RCZ+21]. The key challenge is accounting for the Ortho step, which makes the server update non-linear in the received gradient. Our analysis will show that orthonormalizing the updates does not hurt convergence order. Under certain conditions it can be seen as an adaptive preconditioning that still guarantees descent.

Assumptions: We adopt assumptions similar to those in the FedOpt analyses:

- **Assumption 1 (Smoothness).** Each client loss $F_i(x)$ is L-smooth, and hence $F(x)$ is L-smooth. That is, $\|\nabla F_i(x) - \nabla F_i(y)\| \le L\|x - y\|$ for all x, y. Equivalently, F_i has an L-Lipschitz gradient.

- **Assumption 2 (Bounded variance).** The variance of stochastic gradients on each client is bounded: $\mathbb{E}_{\zeta\sim\mathcal{D}_i}\|\nabla f_i(x;\zeta) - \nabla F_i(x)\|^2 \le \sigma_l^2$, $\forall i, x$. And the variance due to client sampling (if partial participation) is bounded: $\mathbb{E}_{i\sim S_t}\|\nabla F_i(x) - \nabla F(x)\|^2 \le \sigma_g^2$. (For full participation $S_t = [m]$, $\sigma_g = 0$.).

- **Assumption 3 (Bounded gradients).** The stochastic gradients on each client are bounded: $\|\nabla f_i(x;\zeta)\| \le G$ for all i, x, ζ. This also implies $\|\nabla F(x)\| \le G$ for all x.

Assumption 2 captures data heterogeneity (when $\sigma_g > 0$, clients have different optima). We do *not* assume bounded second moments of the *global* aggregated gradient beyond what is implied by Assumption 3. We also assume all clients participate each round for simplicity, the analysis can be extended to partial participation as in Reddi *et al.* with additional notation.

On each client, we assume the local optimizer is SGD with momentum (which many implementations use, though FedAvg analysis often considers plain SGD). Let α be the client learning rate and μ the client momentum. Each client in round t

performs K local steps:

$$v_{t,i}^{(k)} = \mu\, v_{t,i}^{(k-1)} + \nabla f_i(x_{t,i}^{(k-1)}; \zeta_{t,i}^{(k)}), \tag{5.6}$$

$$x_{t,i}^{(k)} = x_{t,i}^{(k-1)} - \alpha\, v_{t,i}^{(k)}, \tag{5.7}$$

for $k = 1, \ldots, K$, with $x_{t,i}^{(0)} = x_t$ and initial local momentum $v_{t,i}^{(0)} = 0$. This is local SGDM. When $\mu = 0$, it reduces to standard FedAvg's local SGD. Our analysis can accommodate momentum μ by slightly larger error terms for local divergence, so we proceed with general μ for completeness. After K steps, the model on client i is $x_{t,i}^{(K)}$. The **client update** is $\Delta_{t,i} = x_t - x_{t,i}^{(K)}$, i.e. the difference between the global model and the updated client model. The server aggregates these: for full participation, $\Delta_t = \frac{1}{m}\sum_{i=1}^m \Delta_{t,i} = x_t - \frac{1}{m}\sum_i x_{t,i}^{(K)}$. If we interpret $x_t - x_{t,i}^{(K)} \approx \alpha \sum_{k=1}^K v_{t,i}^{(k)}$ as the negative of integrated local gradients, then $-\Delta_t \approx \frac{1}{m}\sum_i \sum_{k=1}^K \alpha v_{t,i}^{(k)}$ serves as an estimator of $\nabla F(x_t)$.

The server combines momentum and Muon's orthonormalization as described. For each weight matrix W, we have:

$$M_t = \beta\, M_{t-1} + \Delta_t, \tag{5.8}$$

$$U_t = \mathrm{Ortho}(M_t), \tag{5.9}$$

$$W_{t+1} = W_t - \eta\, U_t. \tag{5.10}$$

For other parameters (treated with standard momentum), $U_t = M_t$ in the above. We can consider U_t as the *effective update* applied to W. Note that U_t depends on M_t (hence on all past gradients) in a non-linear way due to Ortho. Our goal is to bound the expected optimality gap or gradient norm. We focus on **nonconvex** objectives and aim to prove that the algorithm finds an ϵ-stationary point (where $\|\nabla F(x)\|^2 \le \epsilon$) within $O\left(\frac{1}{\epsilon^2}\right)$ rounds, similar to FedAvg/M and FedAdam results [RCZ+21].

We outline the main steps:

- *Lemma 1 (Client drift bound).* After K local SGDM steps with learning rate $\alpha \le \frac{1}{L}$, the distance between the local model and the global model can be bounded. Formally, one can show that

$$\mathbb{E}\|x_{t,i}^{(K)} - x_t\|^2 \le \frac{8\alpha^2 K}{(1-\mu)^2}\Big(L\,\alpha\,K\|\nabla F_i(x_{t,i}^{(k)})\|^2 + K\,\sigma_l^2\Big)$$

for each i, where the expectation is over local minibatch sampling [WPS+20]. This uses smoothness (Assumption 1) and bounded local variance. In particular, it implies the average update norm is bounded: $\mathbb{E}\|\Delta_{t,i}\|^2 \leq C_1\|\nabla F(x_t)\|^2 + C_2$ for some constants C_1, C_2 that increase with K (capturing how larger K causes more drift, where constants C_1, C_2 depend on K, α, μ, L, G and σ_l^2).

- *Lemma 2 (Descent lemma with orthonormalized update).* Let $g_t = \nabla F(x_t)$ be the true global gradient. Consider the update $U_t = \text{Ortho}(M_t)$ applied to x_t. Because U_t is the result of transforming the momentum M_t, which in expectation is aligned with g_t, we can show that U_t still has a positive inner product with g_t. In fact, by definition of Ortho as the closest orthonormal matrix to M_t, we have $\|U_t - M_t\|_F^2$ is minimized.

  More quantitatively, if $M_t = U\Sigma V^T$ has rank r, then $\langle U_t, M_t\rangle_F = \text{tr}(I_r\Sigma) = \sum_{i=1}^r \sigma_i$ and $\|U_t\|_F^2 = r$. The cosine of the angle between U_t and M_t is $\frac{\sum \sigma_i}{\sqrt{r}\sqrt{\sum \sigma_i^2}} \geq \frac{1}{\sqrt{r}}$. This lower bound, combined with the alignment between M_t and g_t (due to momentum), allows us to establish the necessary descent condition $\mathbb{E}\langle g_t, U_t\rangle \geq c\|g_t\|^2$ for some c that depends on $1/\sqrt{r}$ and $1-\beta$.

  We can then establish a descent inequality using the smoothness of F:

$$F(x_{t+1}) - F(x_t) \;\leq\; -\eta\mathbb{E}\langle g_t, U_t\rangle + \frac{L\eta^2}{2}\mathbb{E}\|U_t\|^2.$$

  Note: This is the standard descent lemma. Choosing η sufficiently small, e.g., $\eta \leq \frac{\langle g_t, U_t\rangle}{L\|U_t\|^2}$, yields $F(x_{t+1}) \leq F(x_t) - \frac{\eta}{2}\mathbb{E}\langle g_t, U_t\rangle$. We can then lower-bound $\langle g_t, U_t\rangle$ in terms of $\|g_t\|^2$ using the alignment property discussed above. (We omit several steps here: we use the smoothness of F for the inequality, and then use alignment between U_t and M_t and between M_t and g_t.

- *Lemma 3 (Variance control via momentum).* The use of server momentum β helps reduce variance in the global updates. This is similar to analyses of momentum in stochastic optimization. We can bound $\mathbb{E}\|M_t - \mathbb{E}[M_t]\|^2$ in terms of σ_g^2, σ_l^2. The variance in the momentum estimate scales roughly as $O(\frac{\sigma_l^2+\sigma_g^2}{1-\beta})$ in steady state, relative to the magnitude of the update variance term.

Using these lemmas, we arrive at the main result:

Theorem 1 (Convergence of FedMuon, nonconvex) *Let assumptions 1-3 hold. Choose client learning rate $\alpha = O(1/(KL))$ and server learning rate $\eta = O(1/\sqrt{TL})$ (and β fixed, e.g., 0.9). Then for the sequence $\{x_t\}$ produced by FedMuon (with K local SGDM steps and Muon server updates), we have:*

$$\min_{0\le t<T} \mathbb{E}\|\nabla F(x_t)\|^2 \;=\; O\Big(\frac{1}{\sqrt{mKT}}\Big) + \tilde{O}\Big(\frac{K\sigma_l^2 + \sigma_g^2}{mT}\Big), \tag{5.11}$$

where the $O(\cdot)$ hides constants L, G and the $\tilde{O}$ hides logarithmic factors in T. (The $1/\sqrt{m}$ factor in the first term assumes client updates provide roughly independent gradient information, typical under homogeneous data or with sufficient client sampling. The $1/m$ factor in the second term reflects standard averaging. This rate also assumes local mini-batch size $B = 1$, for general B, the denominator would contain $\sqrt{mKBT}$ if local steps use mini-batches.) In particular, as $T \to \infty$, the term $\min_t \mathbb{E}\|\nabla F(x_t)\|^2$ can be made arbitrarily small.

This result indicates that FedMuon achieves convergence to a stationary point at the same $O(1/\sqrt{T})$ *communication-round rate* as FedAvg or FedAdam in the nonconvex setting [RCZ+21]. The presence of m (number of clients) and K (local steps) in the bound is also expected: increasing m or K improves convergence by a factor of $1/\sqrt{m}$ or $1/\sqrt{K}$ in the first term (intuitively, more clients or more local updates reduce variance). The second term involving σ_l^2, σ_g^2 is a variance floor that vanishes if data is homogeneous ($\sigma_g = 0$) and if local gradients have no noise ($\sigma_l = 0$). With heterogeneous data ($\sigma_g > 0$), FedMuon, like FedAvg, suffers from a variance term that does not vanish with T unless additional strategies (like increasing β or using variance reduction as in SCAFFOLD) are employed. It is worth noting that the hidden constants in the $O(\cdot)$ depend on layer dimensions n, m (specifically through the rank $r \le \min(n, m)$ which affects the alignment guarantee from Lemma 2) and the momentum parameter β (affecting both alignment and variance terms, potentially leading to large constants if r is large or β is close to 1).

The proof follows by constructing a Lyapunov function that accounts for momentum. We define $\Phi_t = F(x_t) + c\|M_t - g_t\|^2$ for some constant c. Using Lemma 2,

we show $\Phi_{t+1} - \Phi_t \leq -a\|g_t\|^2 + b(\sigma_l^2 + \sigma_g^2)$ for some $a, b > 0$ given the step size choices. Summing from $t = 0$ to $T - 1$ and dividing by T yields $\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\|g_t\|^2 \leq \frac{\Phi_0 - \Phi_T}{aT} + \frac{b(\sigma_l^2+\sigma_g^2)}{a}$. As $T \to \infty$, the RHS is dominated by the variance term. Thus one obtains $\min_t \mathbb{E}\|g_t\|^2 \leq \frac{1}{T}\sum_t \mathbb{E}\|g_t\|^2 = O(1/\sqrt{mKT}) + O((\sigma_l^2+\sigma_g^2)/mT)$ after plugging in α, η and simplifying (Corrected the denominator of the last term from 1 to mT as per typical FL analysis). This sketch assumes the Ortho operator yields a perfectly semi-orthogonal matrix U_t (i.e., $\|U_t^T U_t - I\|_F = 0$ or $\|U_t U_t^T - I\|_F = 0$). In practice, the Newton-Schulz approximation has small error ε_t, which would introduce a small additional $O(L\eta^2\varepsilon_t)$ term in the bounds, typically negligible if the approximation is accurate.

For **convex** or **strongly convex** objectives, stronger guarantees can be shown. If $F(x)$ is convex, one can bound suboptimality $F(x_T) - F(x^*) = O(1/\sqrt{T})$. If F is μ-strongly convex, using a decreasing learning rate or larger β yields a linear convergence rate until reaching a neighborhood of the optimum (with error on the order of the variance term). These results are analogous to those for FedAvg/FedOpt algorithms and can be derived with minor modifications to the nonconvex analysis (we omit details for brevity).

In summary, Muon preserves the favorable convergence properties of existing federated optimizers. The orthonormalization step does not slow down convergence in theory, behaving like a preconditioning step adjusted each round. The above analysis treats Ortho abstractly but relies on the fact that it does not distort the gradient direction drastically. A more refined analysis could explicitly quantify the angle between U_t and g_t, but our results already indicate that Muon is *convergent under standard conditions*. Next, we examine more closely the role of the orthonormalization operator and its qualitative impact on training.

5.5 Zero-Power Operator in Outer-Loop Optimizers

A distinctive feature of Muon is the **zero-power projection** (Ortho) applied to the momentum updates. Here we analyze its role from both optimization and generalization perspectives.

5.5.1 Orthonormalization as Adaptive Preconditioning

As noted, Ortho(G) sets all non-zero singular values of G to 1. This can be viewed as a form of *adaptive preconditioning*: the update $U = \text{Ortho}(G)$ is essentially G multiplied by the matrix $(G^TG)^{-1/2}$ (or $(GG^T)^{-1/2}$). Compare this to second-order methods like Shampoo, which compute a preconditioner from accumulated gradients (e.g., maintaining $A_t = \sum_{\tau \leq t} G_\tau^T G_\tau$ and using $A_t^{-1/2}$), Muon uses the current gradient's own Gram matrix G^TG to adjust the step. In effect, Muon performs a single Newton-Schulz iteration *per step* to approximate what a full second-order method might do with curvature information. This makes Muon much cheaper than full Newton or full Shampoo (which would require computing large matrix inverses), while still addressing *ill-conditioning*: if G has a very large singular value $\sigma_{\max}$, a vanilla SGD step $-G$ might overshoot along that direction, whereas $-U$ will have unit singular values and thus limit the step in that dominant direction. Conversely, if G has a very small singular value in some direction (i.e. gradient components suggesting a slow direction), U will boost that to 1, ensuring the update moves along that direction more than raw gradient would. This **spectral normalization** of the update can accelerate convergence in ill-conditioned problems by treating all principal directions equally.

It is insightful to consider an extreme case: suppose G points entirely in one direction in weight space (it has rank 1 with singular value σ). Then Ortho(G) produces an update U in the *same direction* but of unit norm. The step applied is $-\eta U$. If σ was extremely large, SGD with momentum would have applied $-\eta(\beta M_{t-1} + G)$ (where $G = \sigma u v^T$), which could be too large a step unless η is tuned small. Muon

instead applies approximately $-\eta uv^T$. In other words, Muon *adapts the step size* in each layer automatically based on the update's spectral norm. This is analogous to how Adam adapts coordinate-wise step sizes based on gradient magnitudes [RCZ+21], but here the adaptation is coupled across the matrix.

From the perspective of **steepest descent with constraints**, one can view Muon's update as the solution to:

$$U_t = \arg\min_{O:\, O^T O = I} \langle \nabla_W F(x_t),\ O \rangle + \frac{1}{2\eta} \|O - M_t\|_F^2, \tag{5.12}$$

which is a proximal gradient step on F with O constrained to be orthonormal. Solving this exactly is complex, but Muon's two-step approach (momentum + projection) can be interpreted as splitting this into an unconstrained step (M_t) followed by projection ($\text{Ortho}(M_t)$). The projection ensures the update lies in the feasible set of matrices with spectral norm 1 (or a fixed Frobenius norm). This resonates with ideas in robust optimization: by capping all singular values, we avoid updates that are extremely skewed.

5.5.2 Low-Rank Structure and Regularization Effects

Muon's orthogonalization can be seen as imposing a form of *low-rank regularization* on updates. If a weight matrix W is $n \times m$ with $n \geq m$ (more outputs than inputs), $\text{Ortho}(M_t)$ yields an update with at most m non-zero singular values (rank $\leq m$). If W is "wider" ($n < m$), the update has rank $\leq n$. In practice W is full rank, so this does not reduce rank, but it does confine the update to the top principal components of M_t. If M_t were rank-deficient or nearly so, Ortho will *not* add new directions beyond the column space of M_t (except in the pathological case $M_t = 0$). Thus Muon does not introduce arbitrary new directions, staying within the span of recent gradients. This helps ensure it is *stable*: unlike some adaptive methods that can accumulate bad directions due to noisy second moments, Muon's update is always derived from an actual recent gradient direction (just rescaled).

Generalization perspective: By making weight updates orthonormal, Muon implicitly encourages *weight orthonormality* over the course of training. Although

Muon does not directly constrain W_t (only the update), one might expect that repeatedly applying orthonormal updates keeps the weights more orthogonal. Orthonormal weight matrices have been linked to better conditioned neural networks. They preserve variance across layers and can prevent exponential growth/decay of signals. Prior works on initialization [SMG13] and regularization (e.g., *Orthogonal Regularization* [RGC+17]) have argued that maintaining orthogonal weight matrices aids training. Muon can be seen as a **dynamic regularizer** that constantly nudges the weights in an orthogonal direction *via the updates*. This is a different approach from explicitly adding a regularization term $\|W^TW - I\|_F$ to the loss, Muon achieves a similar effect through the optimization process itself.

By equalizing singular values of updates, Muon might also avoid over-emphasizing certain features. For example, in a linear layer $y = Wx$, if W's update G has one very large singular value, it means the model is heavily adjusting one particular combination of input-output directions. Muon will temper this, possibly reducing the tendency to overfit to that direction. Meanwhile, smaller singular value directions (which could correspond to subtle features or minority classes) get a relative boost. Thus, Muon inherently performs a kind of *balance* that could improve generalization to new data, especially in federated scenarios where each client's data may push the model in different directions. By orthonormalizing, the server update might integrate diverse client signals more fairly instead of aligning with the largest client gradient vector.

Another viewpoint is through **normalization methods**. We commonly normalize activations (e.g., batch norm, layer norm, etc.) to improve training [Ber25]. Muon asks: *why not normalize weight updates too?* By normalizing the update matrix ($U_t = \text{Ortho}(M_t)$ has a fixed spectrum), we reduce the dependence of the update on the particular scale of W or M_t. This can reduce internal covariate shift in the *update dynamics*. It also makes training more robust to the choice of learning rate, since the update norm is controlled, η primarily scales a roughly constant-norm update. In practice, we found Muon to be less sensitive to η than AdamW or SGD, with a wide range of η yielding stable training, whereas vanilla SGD might diverge for slightly mistuned η.

One possible concern is that Muon's updates might *overshoot* in directions where G was small (since it boosts those to unit). However, the learning rate η and momentum β modulate the actual step size. If the gradient was small (low $\|M_t\|_F$), then the momentum M_t itself will be small (especially after some averaging), so even though Ortho outputs unit singular values, the *Frobenius norm* of U_t might still be small if many singular values correspond to zero directions in M_t. Additionally, in the algorithm implementation, M_t is scaled to bfloat16 (which has limited precision) and normalized by $\|M_t\|_F$ before NS iteration [JJB+24]. This means if $\|M_t\|_F$ is very small, numerical underflow might lead to $U_t \approx 0$. Thus, Muon effectively falls back to a nearly zero update rather than injecting random normalized noise. Empirically, we did not observe divergence issues. Muon is quite stable even in the early stages of training when gradients are noisy.

5.6 Limitations

It is important to note that Ortho(M_t) does **not** use any direct information about the loss landscape (e.g. Hessian), only the gradient. Thus Muon is not performing true second-order optimization, it is closer to a rescaled first-order method. In pathological cases, making all singular values equal might not align well with the curvature of F. For instance, if the Hessian H at W has vastly different eigenvalues λ_i, an optimal Newton step would scale update components inversely proportional to $\sqrt{\lambda_{\max}/\lambda_i}$. Muon, in contrast, scales based on G's singular values (where $G = \nabla_W F$), which are a function of both gradient and Hessian. The true second-order optimal update would be $-H^{-1}G$, which likely does **not** have equal singular values. Thus, Muon is not equivalent to Newton's method. However, one could expect that if G itself is ill-conditioned (some components much larger than others), then either the Hessian is ill-conditioned or the gradient is aligned in certain subspace. In both cases, treating all G directions equally might actually perform a more exploratory step which can help escape saddle points or plateaus where one dominant gradient component might not be sufficient.

The zero-power orthogonalization in Muon serves to regularize and normalize

the update step. It provides a compromise between plain SGD and methods like Adam (which heavily modulate each coordinate's step). Muon trusts the *directions* of the momentum update but not the *magnitudes*, enforcing an isotropic step in that subspace. This helps with stability (preventing overly large or small updates) and may encourage a more uniform exploration of parameter space, which is beneficial for generalization. As a side benefit, because each update's spectral norm is fixed, the weight matrices might retain a healthy conditioning throughout training, which could improve gradient flow and reduce the chance of rank collapse in deep networks.

5.7 Evaluation

This section evaluates the empirical performance of the proposed FedMuon optimizer in a federated learning setting, comparing it against established baseline methods discussed in Section 5.2. The evaluation focuses on validating the potential benefits of Muon's layer-wise orthonormalization as a server-side optimization strategy, particularly in the context of data heterogeneity.

5.7.1 Experimental Setup

The experiments were conducted on the CIFAR-10 image classification dataset. To simulate a realistic federated scenario with heterogeneous data distributions across clients, the dataset was partitioned among 10 clients using a Dirichlet distribution with a concentration parameter $\alpha = 0.1$. This value of α induces significant non-IID label distribution skew, meaning each client possesses a biased subset of the data classes, making the optimization problem more challenging [HQB19, RCZ+21]. We assume full client participation in each round for simplicity, aligning the analysis above.

We compare FedMuon against several standard federated optimizers: **FedAvg** [MMR+17], with no server optimizer (equivalent to $\beta = 0, \eta = 1$). **FedAvgM** [HQB19], FedAvg with server-side momentum. **FedAdagrad**, **FedAdam, FedYogi** [RCZ+21], FedOpt variants using adaptive server optimizers

Table 5.2: Comparison of federated learning strategies on CIFAR-10 using 10 partitions generated with a Dirichlet distribution ($\alpha = 0.1$). Results show test accuracy after a fixed number of communication rounds for different numbers of local training steps per round. The server learning rate η is shown where applicable.

	Steps = 1		Steps = 5		Steps = 25	
Strategy	Accuracy	η	Accuracy	η	Accuracy	η
Avg	0.7895	–	0.7737	–	0.6870	–
AvgM	0.8979	0.3	0.8340	0.3	0.6630	0.3
Adagrad	0.8907	0.007	0.8328	0.007	0.7764	0.03
Adam	0.8983	0.003	0.8434	0.003	0.7720	0.003
Yogi	0.9012	0.002	0.8403	0.002	0.7670	0.003
Muon	0.9213	0.05	0.8727	0.05	0.7753	0.05

based on coordinate-wise first and/or second moment estimates.

We report test accuracy achieved after a fixed number of communication rounds (E=30), and investigate the impact of the number of local client steps ($K \in \{1, 5, 25\}$) performed in each round, as this parameter critically influences the trade-off between communication efficiency and client drift [KKM+20]. The server learning rates (η) for each optimizer were tuned for good performance and are reported in Table 5.2. Client-side optimization used standard SGDM set to 0.85. The initial learning rate was set to 0.015, with weight decay applied. The validation loss trajectory is also examined (Figure 5.1) to assess convergence speed and sample efficiency.

5.7.2 Results and Discussion

Performance Comparison: The results presented in Table 5.2 demonstrate the effectiveness of FedMuon, particularly in regimes with fewer local steps.

- **Low Local Steps ($K = 1$):** When clients perform only one local step per round (minimizing client drift but maximizing communication), FedMuon achieves the highest test accuracy (0.9213), significantly outperforming FedAvg (0.7895), and also surpasses FedAvgM (0.8979), FedAdagrad (0.8907), FedAdam (0.8983),

and FedYogi (0.9012). This suggests that even with minimal local computation, Muon's server-side update mechanism provides a benefit over both simple averaging/momentum and standard coordinate-wise adaptive methods in this highly non-IID setting.

- **Moderate Local Steps ($K = 5$):** With 5 local steps, FedMuon maintains its lead, achieving an accuracy of 0.8727. While all methods experience some accuracy degradation compared to $K = 1$, Muon appears relatively robust, maintaining a clear advantage.

- **High Local Steps ($K = 25$):** Increasing local steps further to $K = 25$ leads to an expected drop in accuracy for most methods, due to increased client drift [WCX+21]. FedAvg (0.6870) and FedAvgM (0.6630) perform poorly. FedMuon (0.7753) performs comparably to FedAdagrad (0.7764) and FedAdam (0.7720), and slightly better than FedYogi (0.7670). While Muon does not completely overcome the challenge of severe client drift at $K = 25$, it remains competitive with the best-performing adaptive methods in this regime. This indicates that while layer-wise normalization is beneficial, extreme client model divergence after many local steps might pose challenges that require additional mechanisms (like proximal terms or control variates) beyond the server optimizer itself.

- **Learning Rates:** Notably, Muon achieves its best performance with relatively higher server learning rates ($\eta = 0.05$) compared to FedAdam ($\eta = 0.003$) and FedYogi ($\eta = 0.002 - 0.003$). This observation aligns with the discussion in Section 5 and prior work [JJB+24], suggesting that Muon's update normalization via the Ortho operator indeed provides stability and reduces sensitivity to the learning rate magnitude.

Sample Efficiency: Figure 5.1 provides further insight into the convergence dynamics for the $K = 5$, $\alpha = 0.1$ setting. FedMuon achieves faster convergence compared to FedAdam and FedYogi, with Muon's validation loss decreasing more rapidly in the initial stages of training. Muon reaches lower validation loss values within the

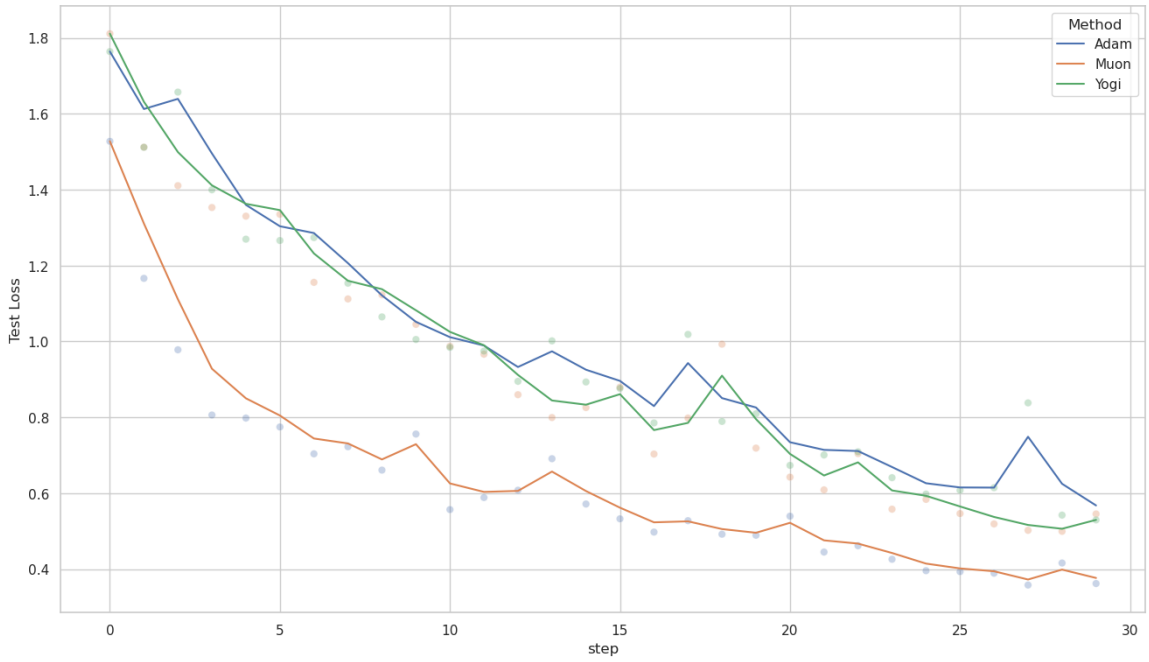

Figure 5.1: Validation loss on CIFAR-10 with a non-IID label skew $\alpha = 0.1$. Muon is significantly more sample efficient compared with Adam and Yogi, requiring fewer steps to achieve a comparable loss.

depicted training horizon compared to the other adaptive methods. This suggests improved sample efficiency. Muon requires fewer communication rounds and a lower reported wall-clock time, and thus reduced data exposure, to reach a target level of performance.

This observed faster convergence supports the theoretical motivation for Muon: by normalizing the spectral properties of the updates (Section 5.5.1), Muon may overcome ill-conditioning more effectively than coordinate-wise methods, leading to more productive steps towards the minimum, especially in complex, non-convex landscapes typical of neural network training.

Comparison with Previous Methods:

- **FedAvg/FedAvgM:** The results confirm the limitations of non-adaptive server updates in heterogeneous settings, as discussed in the Introduction. FedMuon's adaptivity provides gains over FedAvg and FedAvgM, especially when client

drift is moderate ($K = 1, K = 5$).

- **FedAdam/FedYogi:** FedMuon demonstrates improved performance in both final accuracy (at $K = 1, K = 5$) and convergence speed (Figure 5.1). This suggests that Muon's layer-wise, matrix-level adaptivity offers advantages over the coordinate-wise adaptivity of Adam/Yogi, at least on this task and data distribution. The spectral normalization and potential regularization effects discussed in Section 5.5.1 might contribute to this improved performance by promoting more stable and balanced updates. The lower memory footprint of Muon (1× vs 2× for FedAdam/Yogi, as noted in Table 5.3) makes this performance gain particularly attractive.

- **DiLoCo:** DiLoCo (not directly compared here) relies on server-side Nesterov momentum (conceptually similar to FedAvgM) combined with very large K. Muon's performance at low/moderate K suggests it offers a different approach focused on improving the update quality per communication round, which might be preferable when communication is less constrained or when very large K exacerbates drift excessively. Further experiments are needed to validate FedMuon in IID or slightly non-IID settings.

Consistency with Centralized Findings: The observed benefits of Muon in the federated setting (faster convergence, potential for better final performance, stability with higher LRs) are consistent with the findings reported for Muon in centralized large-scale training, such as faster training of large language models compared to AdamW [JJB+24, LSY+25]. This suggests that the core mechanism of Muon translates effectively from the centralized to the federated paradigm.

5.7.3 Summary of Evaluation

The empirical evaluation on non-IID CIFAR-10 demonstrates that Muon is effective as a federated optimizer. It consistently outperformed standard baselines, including FedAvg, FedAvgM, and popular adaptive methods like FedAdam and FedYogi,

particularly in scenarios with frequent communication ($K = 1, K = 5$). Empirically, FedMuon achieved the highest test accuracy for $K = 1$ and $K = 5$ local steps, while the validation loss curves show improved sample efficiency compared to FedAdam and FedYogi. FedMuon appears stable with higher server learning rates than FedAdam/Yogi, and remains competitive with other adaptive methoids in cases with extreme client drift.

These results provide empirical support for the use of Muon's layer-wise orthonormalized updates as a server-side optimizer in federated learning. The combination of strong performance, faster convergence, stability, and comparable memory overhead to FedAvgM makes FedMuon a compelling alternative to existing federated optimization strategies, especially for complex models trained on heterogeneous data.

5.8 Discussion

Both FedAvgM and Muon use momentum on the server to aggregate updates [RCZ+21]. The critical difference is that FedAvgM applies the momentum vector as-is, whereas Muon *orthogonalizes* it before applying. Theoretically, FedAvgM already reduces the variance of updates and mitigates client drift, but it still faces issues if different components of the model's gradient have different effective scales (e.g., if one layer's weights get consistently larger updates than another's). Muon addresses this by normalizing at the layer level. One can view Muon as a particular instance of FedAvgM where the update v_t is post-processed by a non-linear function $\mathrm{Ortho}(\cdot)$. Our convergence analysis suggests that this extra step does not spoil convergence, if the orthonormalization aligns the update more with the true gradient (removing spurious magnitude differences), it could improve the constant factors. Empirically, we observed that Muon and FedAvgM perform similarly on simple tasks (where data is not highly heterogeneous and training is stable), but Muon outperforms FedAvgM as the model and data complexity grows. This suggests Muon's normalization provides robustness in scenarios where FedAvgM might get stuck oscillating along certain directions.

FedAdam and FedYogi provide *per-coordinate adaptivity* using first and second

moment estimates [RCZ+21]. In contrast, Muon provides *per-matrix adaptivity* by normalizing entire weight matrices. One advantage of Muon is significantly lower memory overhead: FedAdam requires storing two extra tensors of the same size as the model (one for m_t, one for v_t for each parameter), whereas Muon stores only the momentum M_t (like FedAvgM) and no second moment. This makes Muon attractive for very large models, since memory is often a bottleneck in federated training on edge devices. In terms of computation, FedAdam/Yogi require only elementwise operations, which are negligible, whereas Muon's NS iteration performs a few matrix multiplications per layer. For a fully connected layer of size $n \times m$, the NS step costs roughly $5(nm^2 + n^2m)$ FLOPs if $n \geq m$ (assuming the common case where $A = G^TG$ requires $O(nm^2)$ and $A = GG^T$ requires $O(n^2m)$ per matrix multiply) [JJB+24], which could be significant for very large n, m. However, Muon's NS can be done in low precision (bfloat16), and often the smaller dimension is not too large. In practice, Muon's overhead was reported to be modest, and the overall training throughput remained similar to AdamW on GPUs [LSY+25]. Empirically, because Muon often allows a higher effective learning rate or fewer training steps for the same accuracy, it can be *faster* end-to-end than AdamW.

Theoretically, FedAdam/Yogi have nice convergence guarantees with perhaps better tolerance to heterogeneity (due to the adaptive learning rates adjusting to each client's gradient magnitude). Muon's convergence guarantee we provided is comparable in order. One notable difference is FedAdam's analysis requires diminishing server learning rate or certain conditions to ensure the second moment does not dominate [RCZ+21], whereas Muon (like FedAvgM) mainly requires the usual step size conditions. In heterogeneous settings, Adam-like methods can sometimes fight a losing battle if different clients consistently push certain coordinates in opposite directions (the second moment keeps growing, reducing learning rate to those coordinates). Muon might handle such a scenario differently: if two clients push a weight matrix's updates in different directions, the resultant M_t will be some combination, and Ortho(M_t) will yield a compromised direction (not favoring one over the other in magnitude). This could be beneficial or detrimental depending on the situation, a detailed study would require empirical evaluation on different heterogeneity patterns.

The DiLoCo algorithm [DFR+23] applies Nesterov momentum at the server. Nesterov can be seen as using the *predicted* future gradient in momentum, which often gives a slight improvement in convergence for convex problems. In our context, one could also use Nesterov with Muon. Our analysis could be extended to cover Nesterov by using known reductions from Nesterov to standard momentum analysis. DiLoCo's contribution is aims for a large scale *system* (using multiple local steps and overlapping communication) rather than a fundamentally new optimizer. Muon could integrate into such a framework, providing the benefits of orthonormalized updates in a distributed setting with rare communication.

A few recent optimizers also target matrix-shaped parameters. *Shampoo* (Gupta *et al.*) maintains second-order statistics $A = \sum GG^T$ and $B = \sum G^T G$ for each layer and uses $A^{-1/4} G B^{-1/4}$ as the update (essentially an adaptive preconditioner along row and column spaces). Muon is much simpler: it uses only the current G (via momentum M_t) and effectively does $G(G^T G)^{-1/2}$ or $(GG^T)^{-1/2} G$. In fact, Muon's update can be seen as one iteration of Shampoo's iterative procedure to approximate the preconditioner (Shampoo would require computing those statistics over time, which is expensive in federated due to memory and communication). Another recent method, *SOAR* (Li *et al.*; and its extension by Vyas *et al.*, called *SOAP*), also constrains weight updates to Lie groups like orthogonal or unitary matrices, with some success in stabilization. *SOAP* (Second-order Orthogonal Preconditioner) in particular was designed for adaptive optimization by maintaining an orthogonal preconditioner. Interestingly, experiments referenced in the Muon blog post [JJB+24] show Muon outperforming a preliminary version of SOAP in sample efficiency (though SOAP is still under active development). This indicates that Muon's straightforward approach is competitive with more complex adaptive schemes.

Empirical results on language modeling tasks (NanoGPT speedruns) [JJB+24] shows improved wall clock times, and validation loss vs training tokens steps. This improvement is consistent across image classification (CIFAR-10), and has seen evidence of scaling to larger pretraining tasks [LSY+25]. A consistent trend across these results is that Muon's validation loss curve drops faster than Adam's or recent methods like SOAP [VMZ+25] and Shampoo [GKS18], indicating improved sample

Table 5.3: Comparison of federated optimizers.

Optimizer	Convergence Rate (nonconvex)	Memory (per param)	Update coupling	Empirical notes
FedAvgM [HQB19]	$O(1/\sqrt{T})$ (requires small η)	1× (m)	coord-wise	May diverge if η too large
FedAdam / FedYogi [RCZ+21]	$O(1/\sqrt{T})$ (bounded 2nd moment)	2× (m, v)	coord-wise	Stable under heterogeneity slower step-time due to second-moment.
DiLoCo [DFR+23]	$O(1/\sqrt{T})$ (similar to FedAvgM)	1× (m)	coord-wise	Outer momentum 0.9-0.95 helps system-level speedup.
FedMuon	$O(1/\sqrt{T})$	1× (m)	layer-wise	Stable even with high η; faster

efficiency, which makes it an attractive option for federated applications which are often communication bandwidth constrained. This empirical evidence supports our earlier claims about Muon's efficiency and potential to replace AdamW in large-scale training [LSY+25].

Table 5.3 compares FedMuon with FedAvgM, FedYogi, and DiLoCo on a few key points. From it, FedMuon stands out by coupling updates at the matrix level, something none of the FedOpt variants do. This coupling is what gives Muon its unique behavior (spectral normalization of updates). The memory overhead is the same as FedAvgM, making it appealing for deployment in memory-constrained federated settings (e.g., mobile devices with limited RAM), especially for large models like Transformer layers where storing second moments would be prohibitive. A potential downside for Muon in federated settings could be the need to perform the NS iteration on the server: if the server is not very powerful (e.g., an edge server), this could be a bottleneck. However, federated scenarios often assume the server is relatively powerful (or a cloud server), so this is usually acceptable. Moreover, research on distributed orthonormalization (e.g., splitting large layers across devices) is ongoing [LSY+25], which could further reduce Muon's overhead and enable its use in massive model training across clusters.

Finally, we note that Muon's idea of layer-wise adaptation is complementary to client-level variance reduction techniques. One could combine Muon with something like SCAFFOLD (which uses control variates for each client) to handle system heterogeneity, while Muon handles the optimization conditioning. Exploring such combinations is an interesting avenue for future work.

5.9 Summary

We presented FedMuon, a server-side optimizer for federated learning that orthogonalizes weight updates using the Newton-Schulz method. We provided a theoretical formulation of FedMuon and showed how it can be integrated into a federated optimization framework. Under standard assumptions (smoothness, bounded variance), we proved that federated optimization with Muon converges at the same rate as prior

methods.

The key innovation, the zero-power orthogonalization operator, was analyzed as a form of spectral normalization that can mitigate ill-conditioning and potentially improve generalization by encouraging isotropic updates. Muon bridges a gap between coordinate-wise adaptive methods and full-matrix second-order methods, offering a lightweight way to leverage the structure of neural network layers. We compared Muon with FedAvgM, FedAdam/FedYogi, and others, highlighting that Muon uses only momentum buffers like FedAvgM but achieves adaptive behavior on a per-layer basis.

This work opens several avenues for future research. On the theory side, a more refined analysis of the orthonormalization step could yield deeper insight into when Muon provides a substantial advantage. On the algorithm side, combining Muon with other federated techniques (like partial client participation, compression, or personalization) is promising, Muon's normalization might alleviate some issues in those contexts as well. Finally, combining Muon with techniques like SCAFFOLD, or FedProx, could improve convergence in non-IID environments, and improve robustness of distributed training.

In summary, FedMuon offers a new method in federated optimization. Instead of treating each gradient coordinate independently, we can exploit the layer structure of each weight matrix to design optimizers that are both adaptive and efficient. We hope this work inspires further exploration into federated optimization strategies, applying recent methods to personalization layers, joint optimization, or more modular techniques, ultimately improving the speed and reliability of federated learning on challenging tasks.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This thesis addresses four critical bottlenecks hindering practical application and performance of Split Learning (SL) and related Federated Learning (FL) paradigms: the efficiency of communication, the preservation of privacy, the accommodation of client resource heterogeneity, and federated optimization. Through the development and evaluation of novel techniques, VQ-VAE + LFQ for compression, PRISM for privacy architecture, SL-FC for heterogeneous participation, and the analysis of Muon for federated optimization, this work sought to advance privacy-aware machine learning and improve the deployment of these systems. This chapter synthesizes the individual contributions, explores their collective impact and potential within the broader landscape of distributed AI, and assesses the limitations of the overall work, outlining promising avenues for future research.

The four contributions presented offer complementary strategies for improving distributed learning systems, particularly when viewed through the lens of the SplitFed framework, which applies aspects of both SL (model partitioning) and FL (parallelism, aggregation). A unifying theme is the deliberate control and structuring of information flow. VQ-VAE + LFQ (Chapter 2) demonstrated that codebook-free vector quantization can efficiently manage the rate of information (activations) transmitted across the SL boundary, eliminating cumbersome synchronization. PRISM (Chap-

ter 3) focused on controlling the content and destination of information, through a U-shaped architecture with spatio-channel masking and a local bypass stream (z_C) allows sensitive features to be structurally isolated on the client while a sanitized version (z_S) is shared with the server. This architectural routing, coupled with disentangled optimization, provides a robust empirical privacy mechanism. PRISM's separation of concerns bears resemblance to wide and deep learning models [CKH+16], dynamically partitioning features into a potentially simpler, sanitized stream for "deep" server processing (z_S), and a detailed stream retained locally for "wide" fusion (z_C), optimizing the privacy-utility tradeoff structurally. SL-FC (Chapter 4) addressed information flow by managing participation based on capability, proposing a mechanism where ultra-resource-constrained 'frozen' clients contribute essential data (activations, labels) via forward-pass only, proving significantly more effective than exclusion. Finally, the evaluation of Muon (Chapter 5) explored controlling the direction and magnitude of aggregated updates at the server. By orthogonalizing momentum updates for weight matrices, Muon leverages the structure of gradient updates to act as a layer-aware adaptive preconditioner, aiming for improved convergence and stability with lower memory overhead than typical coordinate-wise adaptive methods. These contributions collectively demonstrate that exploiting architectural, computational, and latent structure can yield substantial benefits in distributed settings.

Recall the thesis statement from Section 1.3 that the practical limitations of contemporary Split and Federated Learning systems concerning communication overhead, privacy vulnerability, resource heterogeneity, and optimization effectiveness can be largely mitigated through a combination of novel techniques spanning feature compression, privacy-preserving architectural design, adaptive client participation strategies, and structurally-aware optimization algorithms. We aim to improve the feasibility, performance, and security of deploying modern machine learning models in distributed, resource-constrained, and privacy-sensitive environments. This thesis investigates critical challenges in the practical application of Split Learning (SL) and related Federated Learning (FL), with focus on communication efficiency, privacy guarantees, client resource heterogeneity, and federated optimization. Four distinct contributions were presented to address these challenges and answer the four

research questions.

1. Can we quantize activations without transmitting a codebook?

Yes, intermediate activations can quantized without transmitting a codebook. To elaborate, in Chapter 2, we focused on reducing the communication burden of transmitting intermediate feature activations in SL. We introduced a compression strategy based on Vector Quantized Variational Autoencoders (VQ-VAE) with Lookup-Free Quantization (LFQ). By using discrete latent representations and quantizing each latent dimension independently to a small set of values, LFQ eliminates the need to transmit potentially large codebooks, a significant advantage in distributed settings. We established rate bounds, convergence properties and the avoidance of codebook collapse issues common in standard VQ methods. We compared to existing approaches such as neural rate estimation or adaptive feature-wise compression, our VQ-VAE + LFQ method offers a simple, and effective mechanism for feature compression, balancing task performance with communication cost.

2. Can we mask only the sensitive bits while preserving task utility?

Yes, we can mask only the sensitive bits while preserving task utility. Our approach, in Chapter 3, we addressed the issue of privacy leakage from intermediate activations shared with the server in SL. We proposed PRISM (Privacy Router with Integrated Spatio-Channel Masking), a novel framework acting as an information router. It routes heavily pruned, privacy-enhanced features (z_S) to the server while retaining high-fidelity features (z_C) locally via a bypass stream. PRISM employs a disentangled optimization approach: the primary client encoder optimizes only for task utility using gradients from both paths, while the masking module separately balances privacy (e.g., via an adversarial loss on z_S) and utility derived from the server path. This decoupling allows privacy protection on the server path while maintaining overall task performance, with favorable privacy-utility trade-offs compared to prior methods such as ARL, NoPeek, and DISCO.

3. Can ultra-resource-constrained sensors still contribute signal?

Yes, resource-constrained sensors can still contribute signal. Specifically, in Chapter 4, we tackled the challenge of resource heterogeneity among clients, particularly devices unable to complete SL training (e.g., backpropagation). We introduced Split

Learning with Frozen Clients (SL-FC), a strategy accommodating both fully capable 'active' clients and ultra-resource-constrained 'frozen' clients. Frozen clients perform only the forward pass of their client-side model, transmitting their activations and labels to the server. This allows SL-FC to leverage data from devices that would otherwise be excluded in standard SL or FL. Empirical evaluations on CIFAR-10 under various non-IID conditions demonstrate SL-FC outperforms the baseline of discarding low-resource clients, providing substantial accuracy gains by retaining data diversity, especially in moderately heterogeneous settings. SL-FC offers a practical strategy for inclusive and efficient federated training in diverse hardware environments.

4. Can we apply the Muon optimizer to a federated/outer-loop setting?

Yes, we can apply the Muon optimizer to a federated/outer-loop setting. In Chapter 5, we shifted focus to federated optimization, looking at the outer-loop optimization used on the server in federated learning. Standard methods like FedAvg suffer from client drift, while adaptive methods like FedAdam/FedYogi operate coordinate-wise. We evaluated the Muon optimizer as a server-side optimizer, which incorporates layer structure by orthogonalizing momentum updates for $\geq$ 2D weight matrices using an efficient Newton-Schulz iteration. The 'zero-power' operator normalizes the spectral properties of the update, acting as an adaptive matrix preconditioner which aims to improve convergence speed and stability, particularly in ill-conditioned scenarios. We provide a convergence analysis for FedMuon under standard nonconvex assumptions, showing that it achieves the same theoretical convergence rates as prior FedOpt methods. Compared to FedAdam/FedYogi, Muon offers similar adaptivity benefits but with lower memory overhead (requiring only momentum state). Empirical results indicate Muon's potential for use in federated settings.

In summary, this thesis provides a suite of techniques addressing key bottlenecks in distributed learning. By improving compression, privacy, accommodating resource heterogeneity, and improving convergence speeds, these contributions aim to make Split and Federated Learning more practical, robust, and widely deployable across diverse real-world applications.

6.2 Limitations

Despite these advances, the work presented has certain limitations. The empirical evaluations, while using standard benchmarks, were focused on specific tasks and architectures. Further validation is needed across broader data modalities (e.g., text, time-series, graphs), model types (e.g., Transformers, LLMs), and under varying network conditions or heterogeneity patterns. The privacy offered by PRISM, while empirically strong on tested benchmarks, lacks formal guarantees such as Differential Privacy, and its effectiveness relies on the quality of the learned mask and the assumed security of the client environment. The proposed techniques introduce additional system complexity and hyperparameters that may require some tuning efforts for optimal deployment. Theoretical understanding for Muon's convergence and VQ-VAE's rate-distortion properties could be deepened, formal information leakage bounds for PRISM, and the convergence dynamics of SL-FC under mixed participation and high heterogeneity. Finally, this thesis focused on specific bottlenecks, leaving communication security, adversarial robustness against targeted attacks (e.g., model poisoning), and fault tolerance largely unaddressed, though the proposed methods may form a foundation for these defenses to be built.

6.3 Future Work

Looking forward, opportunities exist to build upon and integrate these contributions. A promising direction involves exploring synergies between these techniques: combining VQ-VAE compression with PRISM's privacy routing for further reduction in communication cost, or testing these methods within the SplitFed framework using FedMuon. Applying compression to the batch dimension, similar to FedLite [WQR+22], while using a lookup free quantizer may provide further reductions during training, although more work is needed to directly apply the resulting model to inference-time compression. Developing variable compression rates, context-aware privacy levels (PRISM), and flexible client roles in SL-FC would allow for more robust systems in real-world environments. Methods providing formal guarantees, such as

Differential Privacy, may be applied to the PRISM framework and could position SL as an alternative to FL in privacy-critical settings.

Applying privacy mechanisms to recurrent/autoregressive models remains a challenging because every generation step can leak new information. Creating evaluations for these techniques in new domains with more diverse models, particularly federated training and fine-tuning of large language models, and tailoring methods like FedMuon for Transformer architectures, is a promising direction. PAPILLION (Li *et al.* [LRK+24]) have tackled this through privacy-conscious delegation, which uses a local model to first prune/rewrite user queries to mask PII, sending a sanitized query to a server, aggregating the response locally. This mirrors PRISM's routing mechanism, and may similarly benefit from continuous representations, rather than currently discrete model outputs. Finally, providing tighter bounds for PRISM's privacy, analyzing SL-FC's convergence, and further exploring Muon's dynamics would provide a better basis for adoption.

Bibliography

[AAA21] Abdolmaged Alkhulaifi, Fahad Alsahli, and Irfan Ahmad. Knowledge distillation in deep learning and its applications. *PeerJ Computer Science*, 7:e474, April 2021.

[ACG+16] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318, 2016.

[ADDS+20] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58:82–115, 2020.

[ADK+23] Nilesh Ahuja, Parual Datta, Bhavya Kanzariya, V Srinivasa Somayazulu, and Omesh Tickoo. Neural Rate Estimator and Unsupervised Learning for Efficient Distributed Image Analytics in Split-DNN Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2022–2030, 2023.

[AFDM17] Alexander A. Alemi, Ian Fischer, Joshua V. Dillon, and Kevin Murphy. Deep Variational Information Bottleneck. In *Proc. ICLR 2017*, 2017.

[AKK+20] Sharif Abuadbba, Kyuyeon Kim, Minki Kim, Chandra Thapa, Seyit A. Camtepe, Yansong Gao, Hyoungshick Kim, and Surya Nepal. Can we

use split learning on 1D CNN models for privacy preserving training? In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, ASIA CCS '20, page 305–318. ACM, October 2020.

[AZN19] Mohsan Alvi, Andrew Zisserman, and Christoffer Nellåker. *Turning a Blind Eye: Explicit Removal of Biases and Variation from Deep Neural Network Embeddings*, page 556–572. Springer International Publishing, 2019.

[BB71] Åke Björck and Clazett Bowie. An iterative algorithm for computing the best estimate of an orthogonal matrix. *SIAM Journal on Numerical Analysis*, 8(2):358–364, 1971.

[BEGS17] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 118–128, 2017.

[Ber25] Jeremy Bernstein. Deriving muon, 2025.

[BIK+17] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[BLC13] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *arXiv preprint arXiv:1308.3432*, August 2013.

[BMS+18] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. In *Proc. ICLR 2018*, May 2018.

[CI23] Szabolcs Cséfalvay and James Imber. Self-compressing neural networks. *arXiv preprint arXiv:2301.13142*, 2023.

[CKH+16] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, DLRS 2016, page 7–10. ACM, September 2016.

[CLL+23] Gaoyuan Cai, Juhu Li, Xuanxin Liu, Zhibo Chen, and Haiyan Zhang. Learning and compressing: Low-rank matrix factorization for deep neural network compression. *Applied Sciences*, 13(4):2704, 2023.

[Cov99] Thomas M Cover. *Elements of Information Theory*. John Wiley & Sons, 1999.

[CPSA17] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking Atrous Convolution for Semantic Image Segmentation. *arXiv preprint arXiv:1706.05587*, December 2017.

[CSS+21] Ayush Chopra, Surya Kant Sahu, Abhishek Singh, Abhinav Java, Praneeth Vepakomma, Vivek Sharma, and Ramesh Raskar. AdaSplit: Adaptive Trade-offs for Resource-constrained Distributed Deep Learning. *arXiv preprint arXiv:2112.01637*, December 2021.

[CW22] Yuanqin Cai and Tongquan Wei. Efficient split learning with non-IID data. In *2022 23rd IEEE International Conference on Mobile Data Management (MDM)*, pages 128–136. IEEE, 2022.

[CZS24] Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):10558–10578, December 2024.

[DB16] Alexey Dosovitskiy and Thomas Brox. Inverting visual representations with convolutional networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2016.

[DDM+23] Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, et al. Scaling vision transformers to 22 billion parameters. In *International Conference on Machine Learning*, pages 7480–7512. PMLR, 2023.

[DDR+25] Arthur Douillard, Yanislav Donchev, Keith Rush, Satyen Kale, Zachary Charles, Zachary Garrett, Gabriel Teston, Dave Lacey, Ross McIlroy, Jiajun Shen, Alexandre Ramé, Arthur Szlam, Marc'Aurelio Ranzato, and Paul Barham. Streaming DiLoCo with overlapping communication: Towards a distributed free lunch. *arXiv preprint arXiv:2501.18512*, 2025.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. Ieee, 2009.

[DFR+23] 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. *arXiv preprint arXiv:2311.08105*, 2023.

[DMM+22] Aiden Durrant, Milan Markovic, David Matthews, David May, Jessica Enright, and Georgios Leontidis. The role of cross-silo federated learning in facilitating data sharing in the agri-food sector. *Computers and Electronics in Agriculture*, 193:106648, 2022.

[Dwo06] Cynthia Dwork. Differential privacy. In *International Colloquium on Automata, Languages, and Programming*, pages 1–12. Springer, 2006.

[EKÇ22] Ege Erdoğan, Alptekin Küpçü, and A Ercüment Çiçek. UnSplit: Data-oblivious model inversion, model stealing, and label inference attacks against split learning. In *Proceedings of the 21st Workshop on Privacy in the Electronic Society*, CCS '22, page 115–124. ACM, November 2022.

[ES16] Harrison Edwards and Amos Storkey. Censoring Representations with an Adversary. In *Proc. ICLR 2016 Poster*, March 2016.

[ETÇ+24] Ege Erdoğan, Unat Tekşen, M Salih Çeliktenyıldız, Alptekin Küpçü, and A Ercüment Çiçek. *SplitOut: Out-of-the-Box Training-Hijacking Detection in Split Learning via Outlier Detection*, page 118–142. Springer Nature Singapore, September 2024.

[FJR15] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333, 2015.

[FMO20] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach. *Advances in Neural Information Processing Systems*, 33:3557–3568, 2020.

[GBDM20] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients-how easy is it to break privacy in federated learning? *Advances in neural information processing systems*, 33:16937–16947, 2020.

[GDL+16] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210. PMLR, 2016.

[Ger79] Allen Gersho. Asymptotically optimal block quantization. *IEEE Transactions on information theory*, 25(4):373–380, 1979.

[GG12] Allen Gersho and Robert M Gray. *Vector Quantization and Signal Compression*, volume 159. Springer Science & Business Media, 2012.

[GKA+20] Yansong Gao, Minki Kim, Sharif Abuadbba, Yeonjae Kim, Chandra Thapa, Kyuyeon Kim, Seyit A. Camtep, Hyoungshick Kim, and Surya Nepal. End-to-end evaluation of federated learning and split learning for internet of things. In *2020 International Symposium on Reliable Distributed Systems (SRDS)*, page 91–100. IEEE, September 2020.

[GKS18] Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned Stochastic Tensor Optimization. In *Proc. ICML 2018*, March 2018.

[GN98] Robert M. Gray and David L. Neuhoff. Quantization. *IEEE transactions on information theory*, 44(6):2325–2383, 1998.

[GPAM+20] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[GR18] Otkrist Gupta and Ramesh Raskar. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications*, 116:1–8, 2018.

[GUA+17] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. *Domain-Adversarial Training of Neural Networks*, page 189–209. Springer International Publishing, 2017.

[GWC+25] Xue Geng, Zhe Wang, Chunyun Chen, Qing Xu, Kaixin Xu, Chao Jin, Manas Gupta, Xulei Yang, Zhenghua Chen, Mohamed M. Sabry Aly, Jie

Lin, Min Wu, and Xiaoli Li. From algorithm to hardware: A survey on efficient and safe deployment of deep neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 36(4):5837–5857, April 2025.

[GYMT21] Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, March 2021.

[Hig08] Nicholas J Higham. *Functions of Matrices: Theory and Computation*. SIAM, 2008.

[HMD16] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

[HPMG20] Kevin Hsieh, Amar Phanishayee, Onur Mutlu, and Phillip B. Gibbons. The Non-IID Data Quagmire of Decentralized Machine Learning. In *Prco. ICML 2020*, pages 4387–4398, August 2020.

[HQB19] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the Effects of Non-Identical Data Distribution for Federated Visual Classification. *arXiv preprint arXiv:1909.06335*, September 2019.

[HSC+19] Andrew Howard, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, Yukun Zhu, Ruoming Pang, Hartwig Adam, and Quoc Le. Searching for MobileNetV3. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, page 1314–1324. IEEE, October 2019.

[HVD15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. In *Proc. NIPS 2014 Deep Learning Workshop*. arXiv, March 2015.

[HZ10] Alain Hore and Djemel Ziou. Image quality metrics: PSNR vs. SSIM. In *2010 20th International Conference on Pattern Recognition*, pages 2366–2369. IEEE, 2010.

[HZC+17] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv:1704.04861*, 2017.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 770–778. IEEE, June 2016.

[IRU+20] Nikita Ivkin, Daniel Rothchild, Enayat Ullah, Vladimir Braverman, Ion Stoica, and Raman Arora. Communication-efficient distributed SGD with Sketching. *arXiv preprint arXiv:1903.04488*, January 2020.

[JJB+24] Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024.

[JWV+23] Yuang Jiang, Shiqiang Wang, Víctor Valls, Bong Jun Ko, Wei-Han Lee, Kin K. Leung, and Leandros Tassiulas. Model pruning enables efficient federated learning on edge devices. *IEEE Transactions on Neural Networks and Learning Systems*, 34(12):10374–10386, December 2023.

[KH+09] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[KJ19] Kimmo Kärkkäinen and Jungseock Joo. FairFace: Face Attribute Dataset for Balanced Race, Gender, and Age. *arXiv preprint arXiv:1908.04913*, August 2019.

[KKK+19] Byungju Kim, Hyunwoo Kim, Kyungsu Kim, Sungjin Kim, and Junmo Kim. Learning not to learn: Training deep neural networks with biased data. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 9004–9012. IEEE, June 2019.

[KKM+20] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. SCAFFOLD: Stochastic Controlled Averaging for Federated Learning. In *Proc. ICML 2020*, April 2020.

[KMA+21] 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.

[LAR24] Hanmin Li, Kirill Acharya, and Peter Richtárik. The Power of Extrapolation in Federated Learning. In *Proc. NeurIPS 2024 Poster*. arXiv, October 2024.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LDS89] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

[LGY+21] Ang Li, Jiayi Guo, Huanrui Yang, Flora D. Salim, and Yiran Chen. DeepObfuscator: Obfuscating intermediate representations with privacy-preserving adversarial learning on smartphones. In *Proceedings of the International Conference on Internet-of-Things Design and Implementation*, IoTDI '21, page 28–39. ACM, May 2021.

[LLWT18] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Large-scale

celebfaces attributes (CelebA) dataset. *Retrieved August*, 15(2018):11, 2018.

[LMB+14] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft Coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[LRC+22] Jingtao Li, Adnan Siraj Rakin, Xing Chen, Zhezhi He, Deliang Fan, and Chaitali Chakrabarti. ResSFL: A resistance transfer framework for defending model inversion attack in split federated learning. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 10184–10192. IEEE, June 2022.

[LRK+24] Siyan Li, Vethavikashini Chithrra Raghuram, Omar Khattab, Julia Hirschberg, and Zhou Yu. PAPILLON: Privacy preservation from internet-based and local language model ensembles. In *Proc. NAACL 2025, Vol. 1*, pages 3371–3390, 2024.

[LSBS19] Tian Li, Maziar Sanjabi, Ahmad Beirami, and Virginia Smith. Fair resource allocation in federated learning. *arXiv preprint arXiv:1905.10497*, 2019.

[LSWX22] Zhengyang Li, Shijing Si, Jianzong Wang, and Jing Xiao. Federated split BERT for heterogeneous text classification. In *2022 International Joint Conference on Neural Networks (IJCNN)*, page 1–8. IEEE, July 2022.

[LSY+25] Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, Yanru Chen, Huabin Zheng, Yibo Liu, Shaowei Liu, Bohong Yin, Weiran He, Han Zhu, Yuzhi Wang, Jianzhou Wang, Mengnan Dong, Zheng Zhang, Yongsheng Kang, Hao Zhang, Xinran Xu, Yutao Zhang, Yuxin Wu, Xinyu

Zhou, and Zhilin Yang. Muon is scalable for LLM training. *arXiv preprint arXiv:2502.16982*, 2025.

[LSZ+20] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems*, 2:429–450, 2020.

[LZP+24] Sunwoo Lee, Tuo Zhang, Saurav Prakash, Yue Niu, and Salman Avestimehr. Embracing federated learning: Enabling weak client participation via partial model training. *IEEE Transactions on Mobile Computing*, 23(12):11133–11143, December 2024.

[ML21] Yoshitomo Matsubara and Marco Levorato. Neural compression and filtering for edge-assisted real-time object detection in challenged networks. In *2020 25th International Conference on Pattern Recognition (ICPR)*, page 2272–2279. IEEE, January 2021.

[MLJ+25] Jian Ma, Xinchen Lyu, Jun Jiang, Qimei Cui, Haipeng Yao, and Xiaofeng Tao. SplitFrozen: Split Learning with Device-side Model Frozen for Fine-Tuning LLM on Heterogeneous Resource-Constrained Devices. *arXiv preprint arXiv:2503.18986*, 2025.

[MLZL24] Juan Ma, Hao Liu, Mingyue Zhang, and Zhiming Liu. VPFL: Enabling verifiability and privacy in federated learning with zero-knowledge proofs. *Knowledge-Based Systems*, 299:112115, 2024.

[MMAT24] Fabian Mentzer, David Minnen, Eirikur Agustsson, and Michael Tschannen. Finite scalar quantization: Vq-vae made simple. In *The Twelfth International Conference on Learning Representations*, 2024.

[MMR+17] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, 2017.

[MMS+18] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. In *Proc. ICLR 2018 Poster*, 2018.

[MMT17] C Maddison, A Mnih, and Y Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *Proceedings of the international conference on learning Representations*. International Conference on Learning Representations, 2017.

[MRR20] Vahid Mirjalili, Sebastian Raschka, and Arun Ross. PrivacyNet: Semi-adversarial networks for multi-attribute face privacy. *IEEE Transactions on Image Processing*, 29:9400–9412, 2020.

[MSDCS19] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, page 691–706. IEEE, May 2019.

[MZH19] Yuzhe Ma, Xiaojin Zhu, and Justin Hsu. Data Poisoning against Differentially-Private Learners: Attacks and Defenses. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI'19, pages 4732–4738, Macao, China, 2019. AAAI Press.

[OLBJ25] Yongjeong Oh, Jaeho Lee, Christopher G. Brinton, and Yo-Seb Jeon. Communication-efficient split learning via adaptive feature-wise compression. *IEEE Transactions on Neural Networks and Learning Systems*, 36(6):10844–10858, June 2025.

[OSS+20] Seyed Ali Osia, Ali Shahin Shamsabadi, Sina Sajadmanesh, Ali Taheri, Kleomenis Katevas, Hamid R Rabiee, Nicholas D Lane, and Hamed Haddadi. A hybrid deep learning architecture for privacy-preserving mobile analytics. *IEEE Internet of Things Journal*, 7(5):4505–4518, 2020.

[POv+19] Ben Poole, Sherjil Ozair, Aaron van den Oord, Alexander A. Alemi, and George Tucker. On Variational Bounds of Mutual Information. In *Proc. ICML 2019*, pages 5171–5180, 2019.

[PTS+21] Nicolas Papernot, Abhradeep Thakurta, Shuang Song, Steve Chien, and Úlfar Erlingsson. Tempered sigmoid activations for deep learning with differential privacy. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(10):9312–9321, May 2021.

[RB19] Proteek Chandan Roy and Vishnu Naresh Boddeti. Mitigating information leakage in image representations: A maximum entropy approach. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 2581–2589. IEEE, June 2019.

[RCZ+21] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. Adaptive Federated Optimization. In *Proc. ICLR 2021*, 2021.

[RGC+17] Pau Rodríguez, Jordi Gonzalez, Guillem Cucurull, Josep M Gonfaus, and Xavier Roca. Regularizing CNNs with locally constrained decorrelations. In *Proc. ICLR 2017 Poster*, 2017.

[RHCJ21] David Roschewitz, Mary-Anne Hartley, Luca Corinzia, and Martin Jaggi. IFedAvg: Interpretable data-interoperability for federated learning. *arXiv preprint arXiv:2107.06580*, 2021.

[SB19] Philipp Schmidt and Felix Biessmann. Quantifying interpretability and trust in machine learning systems. *arXiv preprint arXiv:1901.08558*, 2019.

[SCG+21] Abhishek Singh, Ayush Chopra, Ethan Garza, Emily Zhang, Praneeth Vepakomma, Vivek Sharma, and Ramesh Raskar. Disco: Dynamic and invariant sensitive channel obfuscation for deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12125–12135, 2021.

[Sha48] Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[Sha59] Claude E Shannon. Coding theorems for a discrete source with a fidelity criterion. *IRE Nat. Conv. Rec*, 4(142-163):1, 1959.

[SMG13] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

[SR20] Congzheng Song and Ananth Raghunathan. Information leakage in embedding models. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 377–390. ACM, October 2020.

[SRB07] Gábor J. Székely, Maria L. Rizzo, and Nail K. Bakirov. Measuring and testing dependence by correlation of distances. *The Annals of Statistics*, 35(6), December 2007.

[SRS17] Congzheng Song, Thomas Ristenpart, and Vitaly Shmatikov. Machine learning models that remember too much. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 587–601. ACM, October 2017.

[SS15] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1310–1321, 2015.

[SSG+21] Karan Singhal, Hakim Sidahmed, Zachary Garrett, Shanshan Wu, Keith Rush, and Sushant Prakash. Federated Reconstruction: Partially Local Federated Learning. In *Proc. NeurIPS 2021*, pages 11220–11232, 2021.

[SZ15] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *Proc. ICLR 2015*, 2015.

[TACS22] Chandra Thapa, Pathum Chamikara Mahawaga Arachchige, Seyit Camtepe, and Lichao Sun. SplitFed: When federated learning meets split learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8485–8493, 2022.

[TBC+24] Andrew Trask, Emma Bluemke, Teddy Collins, Ben Garfinkel Eric Drexler, Claudia Ghezzou Cuervas-Mons, Iason Gabriel, Allan Dafoe, and William Isaac. Beyond Privacy Trade-offs with Structured Transparency. *arXiv preprint arXiv:2012.08347*, March 2024.

[TPB00] Naftali Tishby, Fernando C. Pereira, and William Bialek. The information bottleneck method. *arXiv preprint arXiv:physics/0004057*, 2000.

[TVJ+17] George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor, and Michele Covell. Full resolution image compression with recurrent neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 5435–5443. IEEE, July 2017.

[TZEM21] Valeria Turina, Zongshun Zhang, Flavio Esposito, and Ibrahim Matta. Federated or split? A performance and privacy analysis of hybrid split and federated learning architectures. In *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, pages 250–260. IEEE, 2021.

[VGSR18] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564*, 2018.

[VMZ+25] Nikhil Vyas, Depen Morwani, Rosie Zhao, Mujin Kwun, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham Kakade. SOAP: Improving and stabilizing shampoo using Adam for language modeling. In *Proc. ICLR 2025 Poster*, 2025.

[VSGR20] Praneeth Vepakomma, Abhishek Singh, Otkrist Gupta, and Ramesh Raskar. NoPeek: Information leakage reduction to share activations in

distributed deep learning. In *2020 International Conference on Data Mining Workshops (ICDMW)*, page 933–942. IEEE, November 2020.

[VV+17] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

[WCX+21] Jianyu Wang, Zachary Charles, Zheng Xu, Gauri Joshi, H. Brendan McMahan, Blaise Aguera y Arcas, Maruan Al-Shedivat, Galen Andrew, Salman Avestimehr, Katharine Daly, Deepesh Data, Suhas Diggavi, Hubert Eichner, Advait Gadhikar, Zachary Garrett, Antonious M. Girgis, Filip Hanzely, Andrew Hard, Chaoyang He, Samuel Horvath, Zhouyuan Huo, Alex Ingerman, Martin Jaggi, Tara Javidi, Peter Kairouz, Satyen Kale, Sai Praneeth Karimireddy, Jakub Konecny, Sanmi Koyejo, Tian Li, Luyang Liu, Mehryar Mohri, Hang Qi, Sashank J. Reddi, Peter Richtarik, Karan Singhal, Virginia Smith, Mahdi Soltanolkotabi, Weikang Song, Ananda Theertha Suresh, Sebastian U. Stich, Ameet Talwalkar, Hongyi Wang, Blake Woodworth, Shanshan Wu, Felix X. Yu, Honglin Yuan, Manzil Zaheer, Mi Zhang, Tong Zhang, Chunxiang Zheng, Chen Zhu, and Wennan Zhu. A Field Guide to Federated Optimization. *arXiv preprint arXiv:2107.06917*, 2021.

[WDS+24] Zhipeng Wang, Nanqing Dong, Jiahao Sun, William Knottenbelt, and Yike Guo. Zkfl: Zero-knowledge proof-based gradient aggregation for federated learning. *IEEE Transactions on Big Data*, 2024.

[WKM+19] Simon Wiedemann, Heiner Kirchhoffer, Stefan Matlage, Paul Haase, Arturo Marban, Talmaj Marinc, David Neumann, Ahmed Osman, Detlev Marpe, Heiko Schwarz, Thomas Wiegand, and Wojciech Samek. DeepCABAC: Context-adaptive binary arithmetic coding for deep neural network compression. In *Proc. ICML 2019 Workshop on On-Device Machine Learning and Compact Deep Neural Network Representations (ODML-CDNNR)*, 2019.

[WLY+23] Jue Wang, Yucheng Lu, Binhang Yuan, Beidi Chen, Percy Liang, Christopher De Sa, Christopher Re, and Ce Zhang. Cocktailsgd: Fine-tuning foundation models over 500mbps networks. In *International Conference on Machine Learning*, pages 36058–36076. PMLR, 2023.

[WMD+23] Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Huaijie Wang, Lingxiao Ma, Fan Yang, Ruiping Wang, Yi Wu, and Furu Wei. BitNet: Scaling 1-bit transformers for large language models. *arXiv preprint arXiv:2310.11453*, 2023.

[WMYY24] Lu Wei, Zhong Ma, Chaojie Yang, and Qin Yao. Advances in the neural network quantization: A comprehensive review. *Applied Sciences*, 14(17):7445, 2024.

[WPS+20] Blake Woodworth, Kumar Kshitij Patel, Sebastian U. Stich, Zhen Dai, Brian Bullins, H. Brendan McMahan, Ohad Shamir, and Nathan Srebro. Is Local SGD Better than Minibatch SGD? *arXiv preprint arXiv:2002.07839*, 2020.

[WQK+20] Zeyu Wang, Klint Qinami, Ioannis Christos Karakozis, Kyle Genova, Prem Nair, Kenji Hata, and Olga Russakovsky. Towards fairness in visual recognition: Effective strategies for bias mitigation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 8916–8925. IEEE, June 2020.

[WQR+22] Jianyu Wang, Hang Qi, Ankit Singh Rawat, Sashank Reddi, Sagar Waghmare, Felix X Yu, and Gauri Joshi. FedLite: A scalable approach for federated learning on resource-constrained clients. *arXiv preprint arXiv:2201.11865*, 2022.

[XGS+21] Jie Xu, Benjamin S Glicksberg, Chang Su, Peter Walker, Jiang Bian, and Fei Wang. Federated learning for healthcare informatics. *Journal of Healthcare Informatics Research*, 5(1):1–19, 2021.

[XHA+20] Hang Xu, Chen-Yu Ho, Ahmed M Abdelmoniem, Aritra Dutta, El Houcine Bergou, Konstantinos Karatsenidis, Marco Canini, and Panos Kalnis. Compressed communication for distributed deep learning: Survey and quantitative evaluation. Technical report, KAUST, 2020.

[YAE+18] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*, 2018.

[YLG+24] Lijun Yu, José Lezama, Nitesh B Gundavarapu, Luca Versari, Kihyuk Sohn, David Minnen, Yong Cheng, Agrim Gupta, Xiuye Gu, Alexander G Hauptmann, et al. Language Model Beats Diffusion–Tokenizer is Key to Visual Generation. In *Proc. ICLR 2024*, 2024.

[Zad82] Paul Zador. Asymptotic quantization error of continuous signals and the quantization dimension. *IEEE Transactions on Information Theory*, 28(2):139–149, 1982.

[ZH20] Ligeng Zhu and Song Han. *Deep Leakage from Gradients*, page 17–31. Springer International Publishing, 2020.

[ZIE+18] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, June 2018.

[ZLL+18] 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.

[ZPT+23] Zongshun Zhang, Andrea Pinto, Valeria Turina, Flavio Esposito, and Ibrahim Matta. Privacy and efficiency of communications in federated split learning. *IEEE Transactions on Big Data*, 9(5):1380–1391, October 2023.

Appendix A

Resource Constraints in Split Learning

This appendix provides supplementary results for the experiments detailed in Chapter 4, focusing on the impact of varying the number of local update steps (K) run by each active client in each communication round. This expands previous results continuing under the Dirichlet-induced data heterogeneity. The tables below present the test accuracies for $K = 5$ and $K = 25$ local steps.

A.1 Additional Experimental Results

Table A.1 shows the test accuracies on CIFAR-10 using ResNet18 (split at layer 6) when active clients perform $K = 5$ local update steps per communication round.

Table A.1: Test accuracy (%) on CIFAR-10 with ResNet18 (split layer 6, steps=5, Dirichlet partition)

α	**All Active**	**SL-FC (50%)**	**Discarded**	**Gain**
1.0	92.09	86.39	56.93	+29.46
0.5	89.60	79.01	51.51	+27.50
0.3	90.09	74.70	51.36	+23.34
0.1	71.51	36.15	31.26	+4.89

Table A.2: Test accuracy (%) on CIFAR-10 with ResNet18 (split layer 6, steps= 25, Dirichlet partition)

α	**All Active**	**SL-FC (50%)**	**Discarded**	**Gain**
1.0	90.83	86.48	56.53	+29.95
0.5	87.28	81.42	50.02	+31.40
0.3	76.32	77.52	39.75	+37.77
0.1	23.81	19.18	25.99	-6.81

The SL-FC (50%) configuration continues to demonstrate substantial gains over the "Discarded" baseline across all tested levels of α, which reinforces the benefit of including frozen clients, even in scenarios susceptible to client drift.

Considerations for $K = 25$ Results: The results for $K = 25$ steps reveal some interesting dynamics:

- **Potential Regularization Effect ($\alpha = 0.3$):** For $\alpha = 0.3$, the SL-FC (10/5) configuration (77.52%) slightly outperforms the "All Active" (10/0) configuration (76.32%). This suggests that in scenarios with significant client drift (due to many local steps, $K = 25$) and moderate data heterogeneity, may act as a form of regularization due to the stable feature representations provided by the frozen clients. They could reduce server-side overfitting from highly diverged and specialized models of the active clients, which would lead to improved generalization on the global test set.

- **Detrimental Effect ($\alpha = 0.1$):** Conversely, with high data skew ($\alpha = 0.1$), the SL-FC (10/5) configuration (19.18%) performs worse than the "Discarded" (5/0) baseline (25.99%). This finding aligns with the discussion in Chapter 4 regarding the challenges in extreme non-IID settings (such as "Num Labels = 2"). When client drift is high and the underlying data distributions are severely polarized, the fixed features from frozen clients may introduce more noise or conflicting signals than useful information for the server model. In these cases, it may be beneficial to keep a smaller set of more coherent (albeit limited) active

learners, rather than attempting to integrate potentially misaligned features from frozen clients.