Technical University of Crete

School of Electrical & Computer Engineering

Diploma Thesis

Federated LoRA-Tuning for LLMs

Author:
Kanakis Kelaidis

Thesis Committee:
Prof. Minos Garofalakis
(Advisor)
Prof. Thrasyvoulos Spyropoulos
Prof. Vasilis Samoladas

A thesis submitted in partial fulfillment of the requirements
for the degree of Diploma in Electrical and Computer Engineering

October 10, 2025

Federated LoRA-Tuning for LLMs

Kanakis Kelaidis

Thesis Advisor: Prof. Minos Garofalakis

Abstract

Large language models (LLMs) have become essential across a wide spectrum of applications, from conversational agents to code generation, making fine-tuning on domain-specific data a ubiquitous need. Yet their deployment in real-world domains is often constrained by data isolation, computational cost and memory requirements. Centralizing proprietary data is frequently infeasible and forcing each organization to train on its own limited dataset typically yields inferior models. Federated Learning offers a natural solution by enabling multiple clients to collaborate without sharing raw data, but naively applying it to massive architectures remains computationally demanding and communication-intensive.

In this thesis, we present a framework for federated fine-tuning of LLMs via Low-Rank Adaptation (LoRA), focusing on efficiency and performance. Building on the recently proposed DP-LoRA framework, we reformulate the original algorithm and aim to evaluate the performance ceiling of federated LoRA-tuning in its non-private form. By introducing small low-rank trainable matrices into transformer attention layers, LoRA reduces the number of tunable parameters by orders of magnitude, making per-client training both feasible and communication-efficient in federated environments. We also implement components for data formatting, inference and parsing for improved data preparation and evaluation, and justify our choice of Gemma3-4B as the backbone model from a plethora of available options. Our experiments compare against the non-private baselines reported in the DP-LoRA study and show that our approach outperforms them, establishing a new benchmark for this setting. These findings highlight the utility of parameter-efficient federated adaptation of LLMs in scenarios where maximizing accuracy and efficiency is the primary goal and suggest promising directions for future research in improving and extending these methods.

Acknowledgments

After five rich and eventful years at the Technical University of Crete, filled with challenges, growth, accomplishments and unforgettable moments, and before setting out on the next stage of my research in the US, I would like to take a moment to express my heartfelt gratitude to everyone who stood by me and helped me reach this point.

First, I would like to thank my supervisor, Prof. Minos Garofalakis, for introducing me to the world of research and pointing me toward interesting directions and niche research areas. I would also like to thank Konstantinos Kechagias from his team at the Athena Research Center for his valuable input and discussions, which contributed to the completion of this thesis.

I am grateful to have met many professors during my time at TUC. I am especially thankful toward Prof. Aggelos Bletsas for his advice and continued consideration, even while he was in the US, and Prof. Thrasyvoulos Spyropoulos for his guidance and encouragement, particularly during my application period. I also appreciate Prof. Daphne Manoussaki, who offered me the opportunity to serve as a teaching assistant. Furthermore, I would like to thank Prof. Athanasios Liavas for his interesting courses in Optimization and Randomized Algorithms, as well as Prof. George Karystinos, whose courses I enjoyed and who was someone I could turn to when administrative matters arose.

However, this experience would not have been the same without the support of my family and close friends. I am grateful to N. Efthimiou, who was perhaps the most influential teacher during my high school years and whose belief in me had a lasting impact on my path. I am thankful to my friends for being by my side throughout these years in Crete, as they made the time pass far more quickly than I expected and turned these years into something truly memorable. Lastly, I want to thank the people I owe the most, my parents, for their unconditional love and for giving me the opportunity to pursue my goals. And of course, my sister - whom I saved for last - for always believing in me and lifting my spirits when I needed it most.

Contents

List of Figures

List of Tables

List of Abbreviations

LLMs Large Language Models

FL Federated Learning

IID Identically and Independently Distributed

HPO Hyperparameter Optimization

DP Differential Privacy

SGD Stochastic Gradient Descent

DP-SGD Differentially Private Stochastic Gradient Descent

NLP Natural Language Processing

PEFT Parameter-Efficient Fine-Tuning

LoRA Low-Rank Adaptation

ICL In-Context Learning

API Application Programming Interface

PATE Private Aggregation of Teacher Ensembles

DP-OPT Differentially Private Optimization

Chapter 1

Introduction

Artificial intelligence has experienced several cycles of promise and stagnation, though the recent rise of deep neural networks and large language models (LLMs) has marked an unprecedented turning point. Today, AI systems are being deployed in a wide range of domains including finance, healthcare, law and education, where they demonstrate substantial impact. These advances are largely driven by the availability of vast datasets and the computational power to train models with billions of parameters.

However, the reality of data availability in practice is often disheartening. Outside of a few industries most domains possess only limited or poor-quality datasets, making the deployment of AI systems more difficult than anticipated. This raises the question of whether organizations could simply pool their data into a common repository for training larger and more accurate models. In practice this is rarely feasible, since data required for AI projects is typically scattered across silos. For example, an AI-driven recommendation system might need product data from retailers, purchase histories from e-commerce platforms, and financial records managed by banks. Each party holds only a partial view, and competitive pressures, privacy concerns and regulatory barriers make it difficult, if not impossible, to merge these sources into a centralized dataset.

Even within a single company integration across departments can face resistance due to internal policies, security requirements and administrative complexity. At a larger scale, the problem of "data islands" is even more pronounced. Attempts to merge sensitive data across research centers, banks or government agencies may run into prohibitive costs and significant legal obstacles. At the same time public awareness of privacy risks has grown, with major companies frequently facing criticism, legal consequences and financial penalties following breaches of user data. As a result, organizations are increasingly reluctant to centralize their most sensitive records despite the promise of better AI models.

Traditional centralized training therefore fails to provide a viable path forward. Large corporations may continue to benefit from massive proprietary datasets, but smaller organizations or those working in regulated domains such as healthcare, finance or law often

cannot acquire sufficient data to train competitive models. This situation poses a pressing challenge: how can we leverage proprietary data distributed across many organizations to advance AI systems, while respecting competition, privacy and security boundaries?

A natural solution is federated learning, which allows multiple parties to collaboratively train a shared model without transmitting raw data beyond its owner's premises. Each participant retains local control over their data while contributing to a global model through aggregation of parameter updates. In this way, FL opens the door to cooperative model training without requiring centralized data integration, enabling cooperation in settings where data sharing would otherwise be impossible.

Yet federated learning is not a silver bullet. It addresses the question of where data should reside, but leaves open another important question of how we can adapt modern architectures, such as billion-parameter LLMs, in a federated setting where not all clients have equal infrastructure capabilities. As we will see in this thesis, while the rise of LLMs has transformed natural language processing, their deployment in real-world environments faces serious challenges due to their large memory and computation demands. These challenges have motivated the development of efficient and practical adaptation strategies.

1.1 Federated Learning as a Natural Framework

Federated learning (FL) is a decentralized paradigm in which multiple clients collaboratively train a shared model while keeping their raw data local. Rather than uploading private records to a central server, each participant computes updates on its own device and transmits only model gradients or parameter deltas [67, 31]. This design parallelizes the workload, since clients perform the bulk of the computation concurrently, and preserves data sovereignty, since proprietary or personal datasets never leave their owners' premises.

Despite these advantages, the federated setting introduces unique challenges. A pronounced example is the presence of non-IID client data, because each client represents a distinct user or organization, their datasets rarely follow the same distribution and are often imbalanced in size. This heterogeneity can lead to several problems like slower convergence or objective inconsistency, where the aggregated global model optimizes a distorted objective compared to the centralized case [62]. Another difficulty arises from communication overhead, naively exchanging full model updates every round becomes infeasible when dealing with billions of parameters.

To address these issues, a variety of communication-efficient FL techniques have been proposed, such as CMFL [42] and ternary compression [66]. Yet even with such methods, the size of modern LLMs makes federated training extremely demanding. This is where parameter-efficient adaptation plays a crucial role, by restricting fine-tuning to a small set of additional parameters, PEFT methods like LoRA reduce both memory consump-

tion and communication overhead, making federated training more practical in real-world deployments. We will return to this synergy between FL and LoRA in Chapter 4.

1.2 Rise of Large Language Models

Large language models (LLMs) have emerged as one of the most transformative advances in artificial intelligence. Systems such as GPT [11], LLaMA [59], and Gemini [55] illustrate how models with billions of parameters, trained on massive amounts of data, can achieve strong results across a wide variety of tasks. Their ability to perform question answering, summarization, dialogue, and reasoning with minimal task-specific supervision has reshaped expectations of what machine learning systems can deliver.

Despite their impressive breadth, however, general-purpose LLMs often underperform in specialized domains. For example, medical applications have motivated models such as Med-PaLM [52], while finance has inspired efforts like BloombergGPT [64]. These examples highlight a critical point that while pretraining on broad datasets provides general knowledge, fine-tuning is required to achieve state-of-the-art performance in domain-specific tasks.

As a result, LLMs are seen not only as academic achievements but also as foundational infrastructure. Their success is further strengthened by the ability to adapt them efficiently and reliably, while taking into account real-world constraints such as limited resources, privacy boundaries, and regulatory restrictions. A more detailed discussion of adaptation strategies will follow in later chapters.

1.3 Toward Parameter-Efficient Adaptation

The most direct way to adapt an LLM is to fine-tune all of its parameters, but this approach is computationally expensive and often infeasible for models with billions of weights. Moreover, in federated environments transmitting full model updates in each round would create prohibitive communication costs. These limitations have motivated the development of lighter adaptation strategies that require updating only a small fraction of parameters while leaving the backbone model frozen.

Among such approaches, parameter-efficient fine-tuning methods have gained particular attention. Techniques such as adapter layers, fine-tuning only the bias terms and Low-Rank Adaptation (LoRA) reduce the adaptation cost by introducing small sets of trainable parameters. LoRA in particular has proven effective, as it allows models to achieve strong domain specialization while substantially reducing overhead, making it a natural candidate for federated learning scenarios. We will return to these methods in Chapter 3 and examine LoRA in more detail in 3.3.4.

1.4 Thesis Organization

In this section we outline the organization of this thesis and its key contributions.

In **Chapter 1**, we describe the landscape that motivates the later chapters, focusing on issues such as data silos and resource limitations in real-world machine learning and discuss how federated learning and parameter-efficient adaptation provide a promising path toward addressing these concerns.

In **Chapter 2**, we explore the domain of Federated Learning. We present its main paradigms (cross-device and cross-silo), describe the standard training workflow, and discuss common challenges such as non-IID client data and communication efficiency.

In **Chapter 3**, we survey Large Language Models. We describe their scaling trends, highlight the attention mechanism of the Transformer architecture, and discuss adaptation strategies with a focus on parameter-efficient fine-tuning, emphasizing LoRA as a particularly relevant method and also comparing closed and open LLMs.

In **Chapter 4**, we present our main contribution, a reformulation of Federated LoRA-tuning without differential privacy. Building upon the DP-LoRA framework, we omit the noise injection mechanism and instead study the performance ceiling of federated adaptation in its non-private form. We introduce our methodological changes, describe the design of our evaluation pipeline and compare our setup against the baselines reported in the original work. In particular, our key contributions are the following:

- We adapt the DP-LoRA algorithm to a non-private setting, providing a more detailed implementation that leaves no vague points.
- We justify the choice of Gemma3-4B as the backbone model from a plethora of options, arguing for its balance between efficiency and performance in federated environments.
- We design and implement data formatting, inference and parsing components for improved data preparation and evaluation in a chat-like format, ensuring clarity across the different datasets.
- We compare our results against the non-private baselines reported in the DP-LoRA study, showing that our choice of model and algorithmic combination achieves stronger downstream performance in the absence of differential privacy. To the best of our knowledge, given the recency of their study, our approach achieves the best results yet reported in this setting.

Finally, in **Chapter 5**, we draw our conclusions, outline the limitations of our approach, and suggest directions for future research on federated adaptation of large language models.

Chapter 2

Federated Learning

2.1 Foundations and Early Notions

Traditional distributed training in datacenters assumes that all data can be centrally collected and then partitioned across compute nodes for parallel processing. This setting, sometimes referred to as datacenter distributed learning [31], focuses on computational efficiency. Parallelism is achieved either by splitting the model across devices (model parallelism) or by distributing training samples across workers (data parallelism). In both cases the training data resides in a single datacenter environment, and communication between nodes happens over reliable interconnections.

By contrast, federated learning (FL) addresses scenarios where the training data is inherently decentralized and cannot be pooled together either for regulatory, privacy or practical reasons. The notion of federated learning was first introduced by McMahan *et al.* [43] in 2016, who explained the terminology as follows:

> "We term our approach Federated Learning, since the learning task is solved by a loose federation of participating devices (which we refer to as clients) coordinated by a central server."

This early definition was mostly connected to the idea of having a very large number of lightweight clients (e.g. smartphones or IoT devices) participating in training, which is now commonly referred to as cross-device FL. It did not explicitly account for the alternative setting where only a small number of powerful organizations (e.g. banks or hospitals) collaborate, which is now called cross-silo FL. To capture both cases under a unified framework Kairouz *et al.* [31] proposed a more general definition:

> "Federated learning is a machine learning setting where multiple entities (clients) collaborate in solving a machine learning problem, under the coordination of a central server or service provider. Each client's raw data is stored locally and

> not exchanged or transferred; instead, focused updates intended for immediate aggregation are used to achieve the learning objective."

In opposition to the mindset of datacenter distributed learning, which primarily aims at parallelism for computational efficiency, the goal of FL is to enable collaboration across silos of data that remain under the control of their owners thereby mitigating systemic privacy risks. This distinction makes federated learning particularly relevant in domains such as healthcare and finance, where sensitive records are distributed across institutions that cannot exchange raw data. At the same time, the reliance on clients connected through heterogeneous and potentially unreliable networks introduces new challenges including communication efficiency, coordination of many clients under limited bandwidth, and the presence of non-IID (identically and independently distributed) and unbalanced datasets.

In the following sections, we discuss the two main paradigms of FL, cross-device and cross-silo, outline the standard training workflow, and highlight common challenges such as non-IID client data and communication overhead.

2.2 Cross-Device vs. Cross-Silo Federated Learning

Federated learning can be deployed under two distinct paradigms depending on the nature of the clients and the environment in which training takes place. While both rely on the same central idea of collaborative learning without raw data exchange, their assumptions, challenges and applications differ substantially.

2.2.1 Cross-Device Federated Learning

Cross-device FL refers to scenarios where training is performed across a very large number of mobile or IoT devices, often reaching populations of millions or even billions of clients (up to 10^{10}) [31]. Each device holds its own local dataset, which never leaves the client, and the training process is coordinated by a central server that never sees raw data. The original study of federated learning by McMahan *et al.* [43] was closely aligned with this setting, motivated by applications such as mobile keyboard prediction or speech recognition, where models improve directly on user devices.

The defining challenges stem from the nature of the clients. Devices are resource-constrained, connected through slow and unreliable networks, and only intermittently available. In practice, most devices participate at most once in training so it is reasonable to assume that each round sees many clients for the first time, this is often referred to as the stateless client assumption. Moreover, only a small fraction of the population can be active in any given round, and even among those selected, 5% or more may drop out due

to low battery, idle state, or poor connectivity [31]. These conditions make communication the primary bottleneck and require algorithms that remain robust under partial participation, device dropouts and highly heterogeneous client data, nevertheless this setting opens opportunities for personalization and on-device intelligence at scale

2.2.2 Cross-Silo Federated Learning

In contrast, cross-silo FL involves a comparatively small number of reliable and resource-rich clients, typically on the order of 2 to 100 organizations, such as banks, research laboratories or medical centers [31]. Each client maintains sizable datasets that cannot be shared directly due to privacy regulations, confidentiality agreements or competitive concerns. Compared to the cross-device scenario, clients in cross-silo FL are typically always online, connected through stable high-bandwidth links, and capable of performing more demanding local training. In this case each client may participate in every round of training, carrying state from one round to the next, which is why they are often referred to as stateful clients.

Moreover, while in the cross-device paradigm data partitioning is always horizontal, meaning clients hold different examples with the same feature space, a distinctive feature of cross-silo FL is that the partitioning of data can vary [31]. In the horizontal case, clients again hold different examples with identical features, as when separate hospitals each maintain patient records. In the vertical case, clients share a common user base but hold disjoint feature sets, as in the case of a bank and a retailer both serving the same customers but with different attributes but each holding different types of information about them (financial compared to purchasing data).

The motivation in this setting is not device-level personalization but rather enabling multiple institutions to train shared models without exposing their proprietary or sensitive data. For instance, hospitals may collaborate to train clinical language models without pooling patient records, financial institutions may jointly develop fraud detection systems without revealing transaction-level data or law firms may work together on specialized legal models. Because the number of clients is limited, system heterogeneity is less pronounced but challenges of non-IID, unbalanced data and communication efficiency remain. These properties make cross-silo FL the natural setting for our work, where we focus on federated adaptation of large language models across institutional data silos.

2.3 Aspects of Federated Training

2.3.1 Federated Learning Training Loop

We now present in more detail the template under which federated learning typically operates. This template encompasses the Federated Averaging (FedAvg) algorithm of McMahan *et al.* [43] and many others, and can be summarized as follows [31]. The training process is orchestrated by a centralized server which repeats the following steps until the training is stopped:

1. **Client selection:** The server randomly samples a number of clients from a population of potential participants. In many applications, participation is subject to eligibility conditions defined by the FL system. These conditions can vary depending on the setting but are usually more pronounced in cross-device FL, for example, a device may only be considered if it is idle, connected to power, and connected via wi-fi.

2. **Broadcast:** The selected clients download the current global model parameters together with training instructions sent by the server.

3. **Client computation:** Each chosen client locally computes a weight update by training on its own dataset. For example by executing SGD on the local data as in Federated Averaging, and in many cases where neural networks are involved this also includes performing backpropagation to update the model parameters.

4. **Aggregation:** The server collects the updates from the clients. For efficiency, stragglers [1] may be dropped once a sufficient number of updates have been received. The server then aggregates the updates, most commonly through a weighted average of client weights as in FedAvg. This stage also serves as an integration point for many other techniques, such as lossy compression to reduce communication and noise addition with gradient clipping to ensure differential privacy.

5. **Model update:** The server updates the global model using the aggregated result, and the process repeats for multiple rounds until convergence.

As noted in [31], the separation of client computation, aggregation, and model update phases is not a strict requirement of FL, and it excludes certain classes of algorithms. For example, asynchronous SGD applies each client's update immediately to the model before any aggregation with others. While such approaches may simplify some aspects of

[1] Stragglers are clients or devices with significantly slower computational or communication speeds that act as bottlenecks, delaying the overall training process.

the implementation of the training system, the synchronous template described above has the advantage of cleanly separating concerns, allowing each component to be studied and improved independently. It is also worth mentioning that in the cross-device paradigm the FL training process should not impact the user experience. During training, the temporary models sent to devices are never used for live predictions, since training configurations may produce poor models. Instead, deployment to users happens only after a separate rollout phase. Likewise, client participation is designed to be invisible to users, running only when devices are idle and connected to power in order not to slow the performance or drain the battery.

Although the training process described above is general and can be applied to many classes of machine learning models, in this thesis we focus on neural networks, and in particular large language models (LLMs). Neural networks are parametric functions composed of multiple layers of transformations that map an input $\mathbf{x}$ to an output $\mathbf{y}$, with trainable parameters adjusted to approximate the underlying function of interest. Training proceeds by minimizing a loss function that quantifies the error between predictions and ground truth labels, using gradient-based optimization (e.g. stochastic gradient descent) together with backpropagation to update the network parameters.

The description above is only a simplified outline of how neural networks are trained in a centralized setting, where the full dataset is accessible and gradients are computed over minibatches drawn from it. In a federated setting, however, data is partitioned across clients, and the training procedure must be adapted. A widely used method for this purpose is the Federated Averaging (FedAvg) algorithm [43]. In FedAvg, each client performs several steps of local SGD on its private dataset and the resulting parameter updates are then averaged by the server to produce the new global model, such an adaptation enables the training of neural networks, including LLMs, under the federated learning framework.

2.3.2 Federated Averaging Algorithm

To formalize the setting, we reproduce the FedAvg algorithm introduced by McMahan *et al.* [43]. Let θ denote the global model parameters and K the number of clients. Each client k has local dataset D_k of size n_k.

Algorithm 1 Federated Averaging (FedAvg)

Require: Number of clients K, clients selected per round C_R, number of communication rounds T, local update steps per client E, local dataset size of client k denoted n_k

Initialize global model parameters $\boldsymbol{\theta}_0$
for each round $t = 1, 2, \ldots, T$ **do**
  Server selects a subset $\mathcal{S}_t$ of C_R clients
  for each client $k \in \mathcal{S}_t$ in parallel **do**
    $\boldsymbol{\theta}_{t+1}^k \leftarrow \textsc{ClientUpdate}(k, \boldsymbol{\theta}_t)$
  end for
  $N_t \leftarrow \sum_{k \in \mathcal{S}_t} n_k$
  $\boldsymbol{\theta}_{t+1} \leftarrow \sum_{k \in \mathcal{S}_t} \frac{n_k}{N_t} \boldsymbol{\theta}_{t+1}^k$
end for

Algorithm 2 ClientUpdate$(k, \boldsymbol{\theta})$

Require: Batch size B, learning rate η, loss function $\mathcal{L}(\theta; x)$, server's weights $\boldsymbol{\theta}$, number of local steps E

Initialize $\boldsymbol{\theta}_k \leftarrow \boldsymbol{\theta}$
for local epoch $e = 1, \ldots, E$ **do**
  $\pi_e \leftarrow$ random permutation of indices of D_k
  $\mathcal{B}_e \leftarrow$ partition $D_k[\pi_e]$ into minibatches of size B
  for each minibatch $b \in \mathcal{B}_e$ **do**
    $\boldsymbol{\theta}_k \leftarrow \theta_k - \eta \nabla_\theta \mathcal{L}(\boldsymbol{\theta}_k; b)$ ▷ optimizer step (e.g., SGD/Adam)
  end for
end for
return $\boldsymbol{\theta}_k$

2.3.3 Hyperparameter Tuning in Federated Learning

Hyperparameter optimization (HPO) has long been a concern in machine learning, and its importance has only grown with the rise of deep neural networks, which are highly sensitive to choices such as learning rate, batch size, optimizer configuration, or regularization strength. In centralized training these values are usually identified through extensive HPO trials, often involving many full training runs. In federated learning, however, this process is considerably more constrained.

When training is performed across resource-limited devices, as in cross-device FL, repeatedly executing long rounds with different hyperparameters can quickly exhaust communication and compute resources. Even in cross-silo FL, where clients are more powerful and reliable, hyperparameter tuning remains costly due to the increased wall-clock time

of each experimental round and the prolonged occupation of high-performance computing units. HPO is even more challenging due to the fact that federated learning introduces additional hyperparameters beyond those present in centralized settings. Examples include the number of clients sampled per round, the number of local steps executed on each client, or the specific form of the aggregation rule at the server. These parameters can significantly affect both convergence speed and final accuracy, and may even determine the quality of the final model [15].

As emphasized in [31], the dimensionality of the hyperparameter search space in FL is therefore much higher, while the cost of evaluation is also greater. This has motivated two research directions: (i) the development of algorithms that are robust to hyperparameter choices, so that a single configuration can perform well across diverse datasets and model architectures and (ii) the design of adaptive or self-tuning optimization methods that reduce reliance on extensive manual search [10, 6].

2.4 Challenges in Federated Learning

Due to its decentralized nature, federated learning faces a variety of challenges. These range from hyperparameter tuning, which we have already discussed, to issues such as robustness, personalization, straggler effects, privacy guarantees, and communication overhead [31]. In this thesis we do not attempt to cover the entire space but instead focus on two key challenges in more detail, namely non-IID client data and communication efficiency , each of which will be addressed in the following subsections.

2.4.1 Non-IID Data

In federated learning, client datasets rarely follow the same distribution and are often imbalanced in size. The most common sources of dependence and non-identicality arise because each client corresponds to a particular user, organization, geographic location or time window [31]. These differences are inherent to the federated setting, in contrast to centralized training where data are typically pooled and shuffled to reach a more uniform distribution.

To make this notion more precise, consider a supervised task with features $\mathbf{x}$ and labels y. A statistical model of federated learning involves two levels of sampling: first sampling a client $i \sim Q$, where Q is the distribution over available clients, and then sampling an example $(\mathbf{x}, y) \sim P_i(\mathbf{x}, y)$ from that client's local distribution. Non-IID data in federated learning typically refers to differences between P_i and P_j for distinct clients i and j. Moreover, both Q and P_i may change over time, introducing temporal drift as another source of non-identicality. Even within a single client, the IID assumption can fail if the data are ordered in a correlated way (e.g. time-ordered text, video frames), though intra-client

correlation can often be reduced through local shuffling.

Taxonomy of non-identical distributions

Rewriting $P_i(\mathbf{x}, y)$ as $P_i(y \mid \mathbf{x})P_i(\mathbf{x})$ and $P_i(\mathbf{x}, y) = P_i(\mathbf{x} \mid y)P_i(y)$ highlights several ways in which client distributions may differ:

- *Feature distribution skew (covariate shift):* The marginals $P_i(\mathbf{x})$ may vary across clients even if $P(y \mid \mathbf{x})$ is shared [2]. For example, in speech recognition, users uttering the same words may produce inputs with different accents, speaking rates or background noise

- *Label distribution skew (prior probability shift):* The marginals $P_i(y)$ may vary across clients even if $P(\mathbf{x} \mid y)$ is the same. For instance, label frequencies often differ across geo-regions, such as in medical domains where certain diseases are endemic to specific regions or in language modeling where some words are more usual in one group than another.

- *Same label, different features (concept drift):* The conditionals $P_i(\mathbf{x} \mid y)$ may vary even if $P(y)$ is shared. The same label may look different across clients due to cultural differences, weather or temporal factors. For example, the label "house" may be associated with snow-covered homes in cold regions and tropical-style homes elsewhere, similarly, the label "greeting" may appear as "hello" in one community and "hi" or "hey" in another.

- *Same features, different label (concept shift):* The conditionals $P_i(y \mid \mathbf{x})$ may differ even if $P(\mathbf{x})$ is shared. In next-word prediction, the same input phrase "How are..." might be completed as "you" by one user and as "things" or "we" by another, depending on personal habits and context.

- *Quantity skew (unbalancedness):* Clients may hold vastly different numbers of samples. For example, a large retail chain may accumulate millions of purchase records, while a small local shop has only a few hundred.

In practice, real-world federated datasets rarely fall clearly into one of the categories above, but instead exhibit a mixture of these effects. The characterization of cross-client differences in partitioned datasets therefore remains an important open question [31]. A clear understanding of which types of heterogeneity dominate in a given application is important for designing effective algorithms.

[2] By "shared" we mean $P_i(\cdot) = P_j(\cdot)$ for all clients i, j

In addition to non-identicality, violations of independence can also arise in federated learning. These occur when the client distribution Q changes over the course of training. A natural example is in cross-device FL, where devices must satisfy eligibility requirements (see 2.3.1) such as being idle, charging and connected to wi-fi before they can participate. Since such conditions are typically met at night local time, participation often follows strong temporal patterns. Because local time of day correlates with geographic location, this introduces geographic bias in the data sampled at each round. Mitigation strategies have been explored in the literature, such as [23].

Furthermore, the presence of heterogeneous and unbalanced client distributions has direct consequences on the optimization process in federated learning. In centralized training, well-shuffled IID minibatches ensure that stochastic gradients approximate the true global gradient. In contrast, when each client samples from its own P_i, client updates may diverge significantly which can slow down convergence or even cause the global model to optimize a mismatched objective function [31]. Specifically, because clients differ in their computing capacities they may perform different numbers of local update steps. Under such heterogeneity, Wang *et al.* [62] showed that FedAvg and related algorithms can converge only to stationary points of a distorted global objective. They termed this phenomenon objective inconsistency and proposed a technique to eliminate the inconsistency problem.

Finally, it is worth noting that most analyses of federated optimization are modeled under an intermittent participation scheme, where in each round C_R clients are sampled to perform local updates out of a total population of K. In cross-device settings, where client availability is unpredictable, it is generally unrealistic to assume $C_R = K$. In contrast, in cross-silo FL, which is the setting most relevant to this thesis, the number of clients is small and reliable, making the assumption that all clients participate in every round more reasonable [31]. This more stable environment simplifies both theoretical analysis and algorithm design.

2.4.2 Communication Efficiency

Communication has been recognized as one of the primary bottlenecks in federated learning. Unlike datacenter training, where high-bandwidth connections are available, federated training relies on wireless or wide-area internet connections that are often slower, less reliable, and more expensive [31]. As model sizes continue to grow, the cost of repeatedly transmitting model parameters between clients and the server can dominate the overall runtime.

Research has explored a range of communication-efficient methods to reduce this overhead. Common approaches include (i) *gradient compression*, where the size of client updates is reduced, for example by quantizing them to fewer bits before being sent to the server, (ii) *model broadcast compression*, which reduces the size of the model transmitted

from the server to clients and (iii) strategies for *local computation reduction*, which can indirectly lessen communication demand. Among these, compressing client-to-server updates typically offers the greatest impact since upload bandwidth is usually more limited than download bandwidth. [31]

In addition, other strategies have been studied, such as federated distillation techniques that exchange model outputs rather than full parameters or wireless co-design methods that exploit channel characteristics for aggregation. Compatibility with privacy-preserving mechanisms like secure aggregation or differential privacy adds another layer of complexity, since aggressive quantization or compression may conflict with these guarantees. [31]

In cross-silo federated learning, communication remains a challenge even though the clients are fewer in number and more reliable, this is primarily due to the increasing size of modern neural networks. For large language models, transmitting billions of parameters in each round can be prohibitive. This motivates the use of parameter-efficient adaptation methods such as LoRA, which substantially reduce the number of trainable and communicated parameters while enabling effective adaptation to downstream tasks. We will return to this point in Chapter 4.

Chapter 3

Large Language Models

This chapter surveys large language models (LLMs), their rapid growth, and the challenges that arise when adapting them to new tasks, and shows how recent parameter-efficient techniques make such adaptation feasible. We begin with an outline of the scaling trends that have led to models with hundreds of billions of parameters, followed by an overview of the Transformer architecture and its attention mechanism, which forms the backbone of modern LLMs. We then review the main families of adaptation strategies, from prompt-based methods to parameter-efficient fine-tuning and full fine-tuning, with a particular focus on Low-Rank Adaptation (LoRA) as a central technique for this thesis. Finally, we compare open- and closed-source LLM ecosystems, highlighting their different implications for adaptation, privacy, and downstream applications.

3.1 Scaling Trends in LLMs

Large language models (LLMs) have become integral to a wide range of applications and have reshaped the landscape of natural language processing (NLP) [70]. Their success builds on the Transformer architecture introduced by Vaswani *et al.* [61], whose fully parallel self-attention layers allow training to scale efficiently on modern GPUs. This parallelism shortens training times and, more importantly, makes it practical to train models with hundreds of billions of parameters.

A sequence of ever-larger Transformer models illustrates the drive toward increasingly massive networks: GPT-2 (1.5 B) [47], GPT-3 (175 B) [11], Gopher (280 B) [48], and Llama2 (70 B) [60], with some models even surpassing 500 B parameters, such as PaLM (540 B) [18] and MT-NLG (530 B) [53]. Studies show that accuracy improves predictably as model size, dataset size and compute grow together [32, 25]. These scaling laws motivate the continued expansion of model capacity, and the widespread adoption of LLMs has made their fine-tuning ubiquitous

However, scaling to today's multi-billion-parameter networks also creates significant challenges. Updating all parameters during fine-tuning is computationally expensive and requires enormous compute and memory resources, while in many domains such as healthcare or finance data also cannot be centralized for training. These limitations have motivated the development of parameter-efficient fine-tuning strategies that adapt models with only a small fraction of additional parameters, but we will return to this topic in a later section. Before that, we turn to the architectural breakthrough that enabled this scaling in the first place, the Transformer and its self-attention mechanism combine efficient parallelization with the ability to capture long-range dependencies.

3.2 Transformer Architecture and Attention

The backbone of modern large language models is the Transformer architecture introduced at [61]. Its central innovation is the self-attention mechanism, which allows each token in a sequence to attend to all others directly. Unlike recurrent networks, which process tokens sequentially, or convolutions, which capture only local dependencies, attention flexibly models both short- and long-range interactions while remaining highly parallelizable on GPUs.

3.2.1 Self-Attention and Multi-Head Attention

Self-Attention Mechanism

Given an input sequence of n tokens represented as embeddings $\mathbf{X} \in \mathbb{R}^{n \times d_{\text{model}}}$, the model computes three projections:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_V,$$

where $\mathbf{W}_Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $\mathbf{W}_K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, and $\mathbf{W}_V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ are learned projection matrices..

Intuitively, queries ($\mathbf{Q}$) encode what each token is looking for, keys ($\mathbf{K}$) encode what each token makes available to others, and values ($\mathbf{V}$) provide the information that is passed forward once attention weights are applied. For example, in the sentence *"The animal didn't cross the street because it was too tired"*, attention helps resolve that "it" refers to "the animal" rather than "the street" [4].

The attention output is then computed as

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}$$

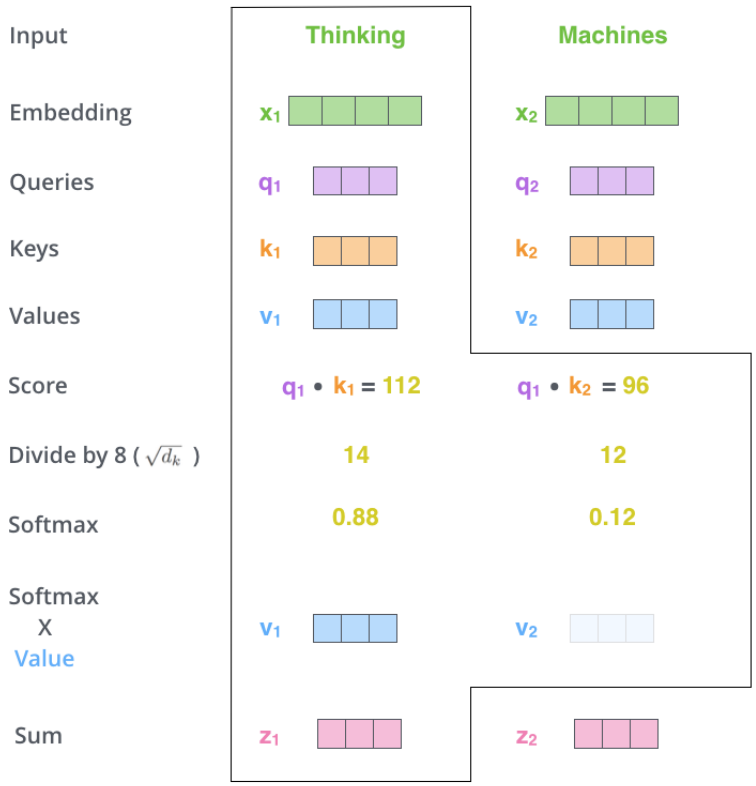

Figure 3.1: Calculation example of self-attention [4]. The score between a query and a key is obtained via their dot product, divided by $\sqrt{d_k}$ to stabilize gradients. In the original Transformer, $d_k = 64$, so scores are divided by 8. The normalized weights are then applied to the values to produce the final representation.

The division by $\sqrt{d_k}$ stabilizes training. To see why, assume the components of $\mathbf{q}$ and $\mathbf{k}$ are independent random variables with mean 0 and variance 1. Their dot product

$$\mathbf{q} \cdot \mathbf{k} = \sum_{i=1}^{d_k} q_i k_i$$

then has mean 0 and variance d_k. Thus, as d_k grows, the variance of the dot products increases proportionally. Dividing by $\sqrt{d_k}$ normalizes this effect and leads to more stable gradients during training [61].

Multi-Head Attention

An expansion of self-attention is multi-head attention, which improves the capacity of the model in two complementary ways.

First, it expands the model's ability to focus on different positions in the sequence. In the earlier example sentence *"The animal didn't cross the street because it was too tired"*,

a single head might capture the link between "it" and "animal", while another could emphasize the relationship between "it" and "tired".

Second, multi-head attention provides multiple representation subspaces. Instead of computing a single set of projections, the Transformer maintains h independent sets of projection matrices. For each head $i \in \{1, \ldots, h\}$, queries, keys, and values are obtained by

$$\mathbf{Q}_i = \mathbf{X}\mathbf{W}_i^Q, \quad \mathbf{K}_i = \mathbf{X}\mathbf{W}_i^K, \quad \mathbf{V}_i = \mathbf{X}\mathbf{W}_i^V,$$

where $\mathbf{W}_i^Q, \mathbf{W}_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ and $\mathbf{W}_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ are learned matrices. Each head computes attention independently as

$$\text{head}_i = \text{Attention}(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i),$$

yielding a d_v-dimensional output.

The outputs of all heads are then concatenated and linearly transformed with $\mathbf{W}^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$:

$$\text{MultiHead}(\mathbf{X}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\, \mathbf{W}^O.$$

This mechanism allows the model to jointly attend to information from different subspaces at different positions, while with only a single attention head these diverse relationships would be averaged together and partially lost. In the original study, $h = 8$ attention heads were used, each learning to capture complementary aspects of the input sequence.

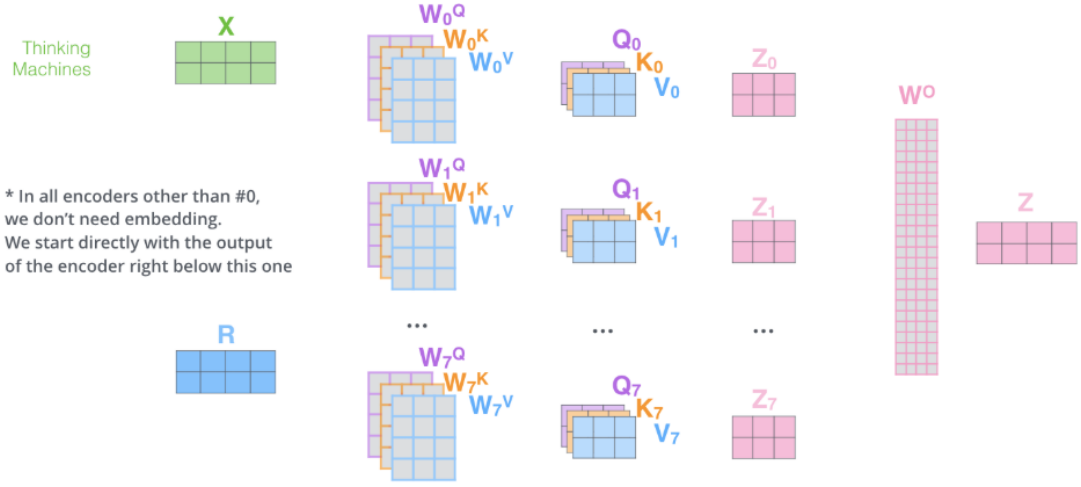

Figure 3.2: Illustration of multi-head self-attention [4]. The procedure consists of: (1) embedding the input sequence, (2) projecting into queries, keys, and values, (3) splitting across multiple heads, (4) computing scaled dot-product attention in each head, and (5) concatenating and projecting with W^O to produce the final output.

3.2.2 Why Attention Matters for Adaptation

The projection matrices $\mathbf{W}_Q, \mathbf{W}_K$, and $\mathbf{W}_V$ constitute a large share of the parameters in a transformer layer and directly determine how information flows between tokens. This is because attention is the mechanism that enables reasoning over context, these matrices are natural points of intervention when adapting a pretrained model to new tasks.

However, updating them in full is prohibitively expensive, with h heads and L layers, each containing multiple dense projections, the parameter count quickly grows into the hundreds of millions even for moderate-size models. This motivates parameter-efficient fine-tuning methods, which restrict training to a much smaller set of additional parameters while keeping the backbone frozen. By targeting the most influential components of the transformer, adapting only this smaller parameter set can still be sufficient to steer the downstream behavior of the LLM.

Low-Rank Adaptation (LoRA) [29] is particularly effective in this setting. It inserts small trainable low-rank matrices into the $\mathbf{W}_Q, \mathbf{W}_K$, and $\mathbf{W}_V$ projections, allowing the model to adjust its attention behavior without modifying the full weight matrices. We return to this mechanism in detail in 3.3.4.

3.3 Adaptation of LLMs

Large language models are generally pretrained on huge amounts of data that cover a wide range of domains, While this pretraining yields broad generalization, in many scenarios users require better performance in a specific domain. For example, a general-purpose model may need to be specialized for financial sentiment analysis or legal document classification. Without adaptation such models often underperform because they are not optimized for the target knowledge domain but instead designed to cover a broad variety of tasks. As a result adapting an LLM to a specific task has become essential, and this adaptation also encompasses alignment with user intent where models are tuned to follow instructions or interact conversationally rather than only predicting the next token.

A variety of strategies for adaptation have been proposed, which can be grouped into three broad families: prompt-based methods, parameter-efficient fine-tuning, and full fine-tuning. Each offers a different balance between efficiency and performance.

3.3.1 Prompt-Based Adaptations

Prompt-based methods steer the behavior of a pretrained model by introducing a small number of additional parameters, typically under 1% of the total weights, while keeping the base model unchanged. The key idea is to provide task-specific prompts that condition the model without changing its actual weights. In the simplest form, these prompts

are manually designed natural language templates, but more effective approaches rely on trainable parameters that can be optimized for a downstream dataset.

One approach is soft prompting, where a sequence of continuous prompt embeddings is prepended to the input tokens and learned during adaptation [38, 37]. Formally, given input embeddings $\mathbf{X} \in \mathbb{R}^{n \times d_{\text{model}}}$, a matrix of trainable prompt embeddings $\mathbf{P} \in \mathbb{R}^{m \times d_{\text{model}}}$ is introduced and concatenated as

$$\mathbf{X}' = [\mathbf{P}; \mathbf{X}]$$

which is then fed into the model.

Another widely used method is prefix-tuning, where a sequence of trainable prompt embeddings is prepended as a prefix to each Transformer layer [33, 35]. In both soft prompting and prefix-tuning, the added prompt embeddings act as a conditioning context that can influence the language model without changing its core parameters. They guide the encoding of the input x by influencing what information is extracted, and they shape the generation of the output y by affecting the next-token distribution. Prefix-tuning extends this mechanism by injecting the context at every Transformer layer rather than only at the input.

These approaches are highly parameter-efficient and effective in low-resource settings, but their expressive power is limited compared to methods that modify deeper layers of the network. As a result, prompt-based methods are well suited for simpler tasks or scenarios where efficiency is the primary concern, but they may fall short in capturing complex domain knowledge or multi-step reasoning.

3.3.2 Parameter-Efficient Fine-Tuning-Based Adaptations

Parameter-efficient fine-tuning (PEFT) methods extend the idea of lightweight adaptation by training only a small fraction of additional parameters while keeping the backbone of the model frozen. This makes them especially popular for adapting very large models, where updating all parameters is not easily possible due to the high cost in memory and computation. This advantage is even more pronounced in federated learning settings, since not all clients have the same infrastructure capabilities, and reducing the number of trainable parameters also lowers the communication cost.

A classical example is adapter layers, where small feed-forward modules are inserted between Transformer blocks and trained for the downstream task while the original network remains unchanged [27]. Another example is BitFit, a simple approach that fine-tunes only the bias terms of the model while freezing all other parameters [68]. The common principle across these methods is that by restricting training to a carefully designed set of parameters, typically less than 10% of the total, one can achieve accuracy close to full fine-tuning while greatly reducing resource requirements.

In practice PEFT methods are particularly well suited for tasks of medium complexity,

since they provide stronger adaptation capacity than prompt-based methods while avoiding the high cost of full fine-tuning. We devote a dedicated subsection to one of the most widely adopted and effective PEFT approaches, Low-Rank Adaptation (LoRA), where we will examine the method in more detail since it plays an important role in the next chapters of this thesis.

3.3.3 Full Fine-Tuning-Based Adaptations

The most straightforward strategy is to update all parameters of the pretrained model for the specific application. In practice, full fine-tuning-based adaptations may either update the entire network or, in lighter variants, only the last few layers. A method of full fine-tuning is gradual unfreezing, where training begins with the last layer, then progressively unfreezes the next lower layer in each step, until all layers are fine-tuned and the model converges [28, 49].

Updating the entire model often provides the strongest possible performance, since every component of the network can adapt to the target task. However, for models with billions of parameters full fine-tuning is computationally and memory intensive. It also requires storing and transmitting the full adapted model which is impractical in distributed or federated environments where communication efficiency is of great importance.

As a result, full fine-tuning is rarely feasible outside well-funded industrial labs, though it remains the most reliable path to achieving the best results when resources are available and the downstream task is sufficiently challenging.

In summary, adaptation methods fall into three major families, each offering a different trade-off between efficiency and performance. Prompt-based methods are the most lightweight but limited in adaptation capacity, full fine-tuning is the most powerful but resource-intensive, and parameter-efficient methods strike a middle ground. In the next subsection we examine LoRA in greater detail, setting the stage for its extension to the federated learning scenario in Chapter 4.

3.3.4 Low-Rank Adaptation

Motivation and Mechanism

A popular parameter-efficient fine-tuning method is *Low-Rank Adaptation* (LoRA) [29]. The motivation arises from the growing difficulty of adapting extremely large pretrained models to multiple tasks. As we previously discussed, while full fine-tuning updates all model parameters and provides strong performance, it requires storing and deploying a full copy of the model per task. Parameter-efficient approaches aim to avoid this issue by training a small number of task-specific parameters on top of the frozen backbone.

However, many existing methods introduce inference latency by extending model depth (e.g. adapters [27, 50]) or reduce the model's usable sequence length (e.g. prefix- or prompt-tuning [35, 33, 38]), and often fail to fully match the performance of fine-tuning.

LoRA is inspired by results showing that learned over-parameterized models actually operate in a space of much lower intrinsic dimension [34, 3]. In particular, Aghajanyan et al. [3] demonstrated that pretrained language models can still learn efficiently even after random projection into a smaller subspace. Motivated by this, the LoRA paper hypothesizes that weight updates during adaptation also have a low *intrinsic rank*, and represents them via a low-rank decomposition. In fact, the authors show that even extremely small ranks (e.g., $r = 1$ or $r = 2$ for GPT-3 layers with $d = 12{,}288$) can suffice to achieve competitive adaptation quality [29].

To see the core idea, consider a standard linear layer,

$$\mathbf{h} = \mathbf{W}_0\mathbf{x}, \quad \mathbf{W}_0 \in \mathbb{R}^{k\times d},\ \mathbf{x} \in \mathbb{R}^d$$

where W_0 is a pretrained weight matrix. LoRA freezes W_0 and constrains its update to a low-rank form:

$$\mathbf{h} = (\mathbf{W}_0 + \Delta\mathbf{W})\mathbf{x}, \quad \Delta\mathbf{W} = \tfrac{\alpha}{r}\mathbf{B}\mathbf{A},$$

with $\mathbf{A} \in \mathbb{R}^{r\times d}$, $\mathbf{B} \in \mathbb{R}^{k\times r}$, and $r \ll \min(d, k)$. During training, only $\mathbf{A}$ and $\mathbf{B}$ are updated, while $\mathbf{W}_0$ remains fixed. The update $\Delta\mathbf{W}\mathbf{x}$ is scaled by α/r, in practice the paper suggests setting $\alpha = r$ so no additional hyperparameter tuning is needed.

Initialization sets $\mathbf{B} = \mathbf{0}$ and $\mathbf{A} \sim \mathcal{N}(0, \sigma^2)$, ensuring that $\Delta\mathbf{W} = 0$ at the beginning of training and the pretrained model output is preserved. This setup is illustrated in Figure 3.3.

Practical Benefits and Applications

The reparameterization shown above reduces the trainable parameters per layer from $O(dk)$ to $O(r(d + k))$. For example, with $d = k = 4096$ and rank $r = 8$, LoRA requires fewer than 0.4% as many trainable parameters as full fine-tuning for that layer. The reduction in trainable parameters translates into several practical advantages. A single pretrained backbone can be shared across many tasks, with each task requiring only a compact pair of matrices $(\mathbf{A}, \mathbf{B})$ that can be stored and swapped efficiently. This greatly lowers the storage requirement and makes task switching almost instantaneous. Furthermore, because the design is purely linear, the trainable matrices can be merged into the frozen weights after training, ensuring that inference runs exactly as in the original model. As a result LoRA introduces no additional latency unlike other methods.

While LoRA can in principle be applied to any dense projection, in practice it is most often used in the self-attention mechanism. The original paper tested its insertion into the query, key, value, and output projections $(\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V, \mathbf{W}_O)$. They reported that

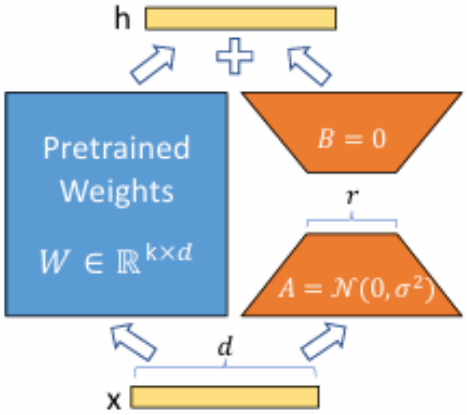

Figure 3.3: Schematic of LoRA applied to a linear layer [29]. The pretrained weight $\mathbf{W}_0$ is frozen, while the update $\Delta\mathbf{W} = \mathbf{BA}$ is parameterized with low rank r. $\mathbf{B}$ is initialized to zero and $\mathbf{A}$ from a Gaussian distribution.

applying LoRA to all four projections achieved almost the same accuracy as applying it to only two ($\mathbf{W}_Q$ and $\mathbf{W}_V$ or $\mathbf{W}_Q$ and $\mathbf{W}_K$), indicating that effective adaptation can be obtained even with partial coverage of the attention module. Although many combinations were explored, the paper did not provide a direct head-to-head comparison between adapting $(\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V)$ and adapting all four $(\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V, \mathbf{W}_O)$. In practice, most implementations therefore adopt $\mathbf{W}_Q, \mathbf{W}_K$, and $\mathbf{W}_V$ as the standard choice. Lastly, LoRA has been shown to outperform other popular PEFT approaches such as BitFit [68] and several variants of adapter tuning [27, 36, 46, 51]. This combination of efficiency and effectiveness has made LoRA a widely used PEFT method and a central component of our work in Chapter 4.

3.4 Open vs. Closed LLMs

In response to the dominance of proprietary systems such as GPT [45], Claude [8], and Gemini [55], a parallel ecosystem of open-weight LLMs like LLaMA [59], Vicuna [16], Mistral [5], and many others has emerged. Although the quality gap is narrowing, closed models still outperform their open counterparts on public leaderboards [17] and require zero infrastructure effort, since the model weights reside on the provider's servers and all interaction happens through remote calls. This convenience keeps them attractive even for tasks involving highly sensitive data.

As discussed in Section 3.3, adaptation plays a crucial role in tailoring LLMs to specific domains. Yet because the weights of closed models are inaccessible, such adaptations cannot fine-tune parameters directly and must instead rely on the construction of

privacy-preserving discrete prompts. However, multiple studies have shown that private information can leak from seemingly benign interactions with LLMs. Query-based attacks can retrieve memorized training data [13, 14], while other work demonstrates vulnerabilities to membership-inference attacks and highlights leakage during in-context learning [20, 21]. In response, a wave of privacy-preserving adaptation methods for closed APIs has emerged. Representative examples include PromptPATE [20] from NeurIPS 2023 and three ICLR 2024 approaches, DP-OPT [26], DP-FewShotGen [54], and DP-ICL [65]. By contrast, open models make their weights fully available, supporting the full spectrum of adaptation strategies and making them indispensable for both research and downstream applications where full fine-tuning or PEFT methods such as LoRA (Subsection 3.3.4) are needed.

3.4.1 Privacy Leak Points at LLMs

We will now analyze the inherent privacy limitations of closed LLMs and discuss why open LLMs offer a more secure and flexible solution. In our comparison, we will use material from [24].

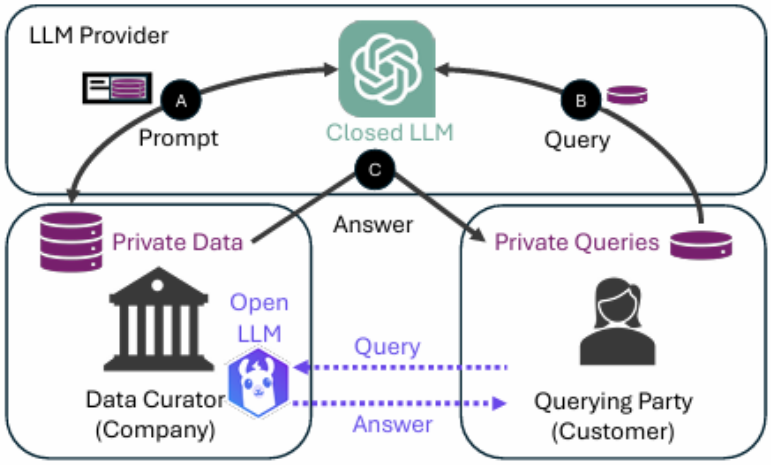

Figure 3.4: Privacy flows in closed vs. open LLMs [24]. Three roles are shown: (1) a provider hosting a proprietary LLM, (2) a curator holding sensitive data, and (3) a querying party submitting private requests. The dashed purple arrows highlight how local adaptation of an open LLM eliminates leaks at **A** and **B**, offering end-to-end protection.

There are three points where private data can leak when using an LLM (see Fig. 3.4). First, during prompt construction (**A**), the data curator must package sensitive records into a discrete prompt, exposing their entire dataset (or at least a part of it) to the provider. Second, when the querying party submits a private request (**B**), that query

is sent unprotected and can be observed by the provider. Third, the LLM's responses can inadvertently reveal curator data to the querying party through the returned answers of the prompted LLM (**C**) [21]. Existing closed-LLM defenses like PromptPATE [20], DP-FewShotGen [54], and DP-ICL [65] only guard against leakage at **C**; none protect against **B**. To prevent leakage at **A**, these methods require access to a powerful local open LLM. As an alternative (dashed purple lines in Fig. 3.4), the data curator can privately fine-tune an open LLM locally and let the querying party interact directly with it, thereby protecting against **A**, **B**, and **C**. Consequently, using an open LLM is unavoidable for complete privacy protection.

According to Figure 3.4 and the experimental results in Hanke *et al.* [24], closed-LLM adaptations inevitably leak users' private queries at inference time (**B**) and expose sensitive records during prompt construction (**A**). By contrast, as illustrated by the dashed purple arrows in the figure, simply adopting an open LLM for local fine-tuning avoids these leak points outright, without the need for additional defenses. Moreover, even when those open models are much smaller, private local fine-tuning consistently outperforms closed-LLM prompt-based adaptations in downstream accuracy. Finally, the combined training and query costs of adapting closed LLMs (API access fees charged by the provider) exceed the expenses of local adaptations on open LLMs (estimated from cloud training and inference costs). Together, these findings show that private, local adaptation of open LLMs is preferable in privacy, utility, and cost.

Chapter 4

Federated LoRA-Tuning

This chapter focuses on our approach to adapting large language models through federated low-rank adaptation (LoRA). We build on the framework proposed in [40]. Our key modification is that we omit the differential privacy component and instead focus on evaluating the effectiveness of federated LoRA tuning combined with our chosen model. The central objective is to understand how this adjustment impacts downstream performance and to compare our results against the baselines reported in the original study. The chapter begins with the motivation behind this choice, followed by a discussion of related work and a detailed overview of the baseline method. We then proceed to outline our own approach, emphasizing the differences from the initial study, and present the implementation details and experimental results that support our findings.

4.1 Motivation

The previous chapters have outlined the landscape of large language models, the role of parameter-efficient adaptation techniques, and the opportunities opened by federated learning. In this chapter we narrow our focus to a concrete instantiation of these ideas, federated low-rank adaptation (LoRA).

As LLMs continue to grow in scale, the need for efficient adaptation methods becomes increasingly urgent. Most practitioners and organizations interested in downstream applications do not have the resources to fully fine-tune models with billions of parameters, since such efforts require enormous computational budgets and expensive infrastructure. Parameter-efficient methods such as LoRA address this limitation by introducing only a small number of additional trainable parameters while keeping the original weights frozen [29]. This dramatically lowers memory and communication costs making fine-tuning more accessible without sacrificing much performance. When combined with federated learning, LoRA further enables distributed data-holders to adapt models collaboratively on private

datasets offering a scalable path toward deploying LLMs in real-world environments.

The importance of such methods is evident in many domains. In healthcare, for example, hospitals may wish to adapt an open LLM to clinical notes or medical records in order to assist doctors with summarization, diagnosis support or patient communication. In finance, banks may want to fine-tune models on proprietary transaction data to improve fraud detection or customer service chatbots. In both cases data cannot easily be centralized due to regulatory, privacy or competitive concerns. Federated learning provides a natural way for enabling collaboration across these silos by exchanging model updates instead of raw data. When used along with LoRA this approach further reduces the burden on local clients, since only lightweight low-rank adapters need to be trained and communicated rather than the full model.

The study we build upon, *Differentially Private Low-Rank Adaptation of Large Language Models Using Federated Learning* [40], argues that combining differential privacy (DP) with federated LoRA provides both privacy guarantees and communication efficiency. While this is a compelling direction for highly sensitive applications, the inclusion of DP noise necessarily trades off accuracy for privacy. In practice, this raises a key question: what is the performance ceiling of federated LoRA-tuning itself independent of any privacy constraints?

This question motivates our work. By removing the DP aspect we aim to evaluate federated LoRA tuning in its purest form, isolating the contributions of parameter efficiency and collaborative training. Doing so allows us to establish an upper bound on the achievable utility of this method family. Comparing our results to the baselines reported in the original DP-LoRA study we highlight how our choice of model and algorithmic combination leads to consistently stronger performance. Our focus is on demonstrating that a carefully selected LLM paired with federated LoRA tuning can surpass the baseline results established in their work.

This perspective is valuable in its own right, since many real-world scenarios do not require formal DP guarantees but still demand scalable, communication-efficient ways of adapting large models across decentralized datasets. In such scenarios, maximizing accuracy and efficiency takes precedence over strict privacy accounting. By focusing on the non-private variant of federated LoRA, we address this broader class of use cases and demonstrate the full potential of the method when privacy is not the primary constraint.

4.2 Related Work

4.2.1 Background and Motivation of DP-LoRA

Liu *et al.* [40] position their work at the intersection of three trends: the growing demand for domain-specific large language models, the increasing adoption of federated learning

in sensitive data environments, and the rising concerns about privacy leakage from model training and inference.

First, while general-purpose LLMs such as GPT-2/3/4 [47, 11, 2], BLOOM [63], Gemini [56] or LLaMA [60] have demonstrated impressive performance on a wide range of tasks, their effectiveness drops when applied to specialized fields such as medicine, finance, and law. This has motivated a wave of domain-specific LLMs, including Med-PaLM for clinical knowledge [52], BioGPT for biomedical literature [41], and BloombergGPT and FinGPT for financial applications [64, 39]. These efforts highlight the value of tailoring models to narrower domains, but they also underscore a key obstacle because the necessary training data in such domains is often sensitive and distributed across multiple.

Second, the federated learning setting provides a natural way to address this distribution. Hospitals, banks, or research labs can collaboratively adapt a model without sharing their raw data instead transmitting only model updates to a central aggregator. This setup respects regulatory and confidentiality constraints while still enabling collective improvements. However, two major challenges exist in applying FL to LLMs. On the one hand, FL by itself does not guarantee privacy. Prior work has demonstrated that model parameters or generated outputs can reveal sensitive training data, enabling attacks such as training data extraction or membership inference [13, 21]. On the other hand, the sheer scale of LLMs introduces large communication overhead since millions or even billions of parameters must be exchanged across clients and server in each round of training.

The DP-LoRA framework is proposed as a solution to these challenges. It combines differential privacy, which provides formal guarantees against information leakage by injecting calibrated Gaussian noise into weight updates with low-rank adaptation, which restricts fine-tuning to a small set of trainable matrices and thus reduces the volume of parameters that need to be transmitted. In this way, DP-LoRA seeks to both preserve privacy and improve communication efficiency, enabling practical federated fine-tuning of LLMs for sensitive, domain-specific applications.

4.2.2 Method Overview DP-LoRA in Federated Learning

We begin by fixing notation for deep neural networks and for the federated setting.

Deep Neural Networks

We begin by fixing notation for deep neural networks and for the federated setting. Let $\mathbf{x} \in \mathbb{R}^d$ be an input vector and $\hat{\mathbf{y}} \in \mathbb{R}^k$ the model output produced by a weight matrix $\mathbf{W} \in \mathbb{R}^{k \times d}$ and a bias $\mathbf{b} \in \mathbb{R}^k$,

$$\hat{\mathbf{y}} \;=\; \mathbf{W}\,\mathbf{x} + \mathbf{b}.$$

The trainable parameters are collected in $\boldsymbol{\theta} = \{\mathbf{W}, \mathbf{b}\}$. A loss function $\mathcal{L}(\boldsymbol{\theta}; \mathbf{x}, \mathbf{y})$ evaluates the difference between the prediction $\hat{\mathbf{y}}$ and the ground truth $\mathbf{y}$. We also assume that the training dataset D contains N labeled pairs and that training proceeds in mini-batches of size B with samples $\{(\mathbf{x}^b, \mathbf{y}^b)\}_{b=1}^{B}$. At each iteration the batch gradient and the parameter θ are updated by the gradient descent method as follows:

$$\begin{aligned} \mathbf{g} &= \frac{1}{B} \nabla_{\boldsymbol{\theta}} \sum_{b=1}^{B} L(\boldsymbol{\theta}; \mathbf{x}^b, \mathbf{y}^b), \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \gamma \cdot \mathbf{g}, \end{aligned}$$

for learning rate γ, over T iterations.

Federated learning

We now extend this setting to distributed training across K clients. Client k holds a private dataset D_k of size N_k; the total sample size is $N = \sum_{k=1}^{K} N_k$. Using a batch of size B on client k induces a local sampling probability $q_k = B/N_k$. A single communication round consists of the following three steps:

1. The server broadcasts the current global model parameters $\boldsymbol{\theta}$ to all K clients.
2. Each client k initializes its local parameters $\boldsymbol{\theta}_k \leftarrow \boldsymbol{\theta}$ and updates them using gradient descent on a mini-batch of B samples from its local dataset D_k.
3. The server collects the updated parameters $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K\}$ from all clients and aggregates them into a new global model:

$$\boldsymbol{\theta} = \sum_{k=1}^{K} \rho_k \cdot \boldsymbol{\theta}_k,$$

where $\rho_k = N_k/N$ is the weight factor, with $\sum_{k=1}^{K} \rho_k = 1$.

The training process repeats these steps for T communication rounds, after which the server obtains the final global model.

Method Overview

The proposed DP-LoRA algorithm follows the general federated learning loop described above. We assume K clients with distributed datasets, each participating in the synchronous update of a shared large language model. At the beginning of training every client holds an identical copy of the model parameters $\boldsymbol{\theta}$. These consist of the frozen weights $\mathbf{W}$ of the pretrained LLM together with the trainable low-rank adapter weights

A and **B**. During local training, each client k keeps the backbone **W** fixed and fine-tunes only its own adapters $(\mathbf{A}_k, \mathbf{B}_k)$. After local updates the adapters are returned to the server, which aggregates them to update the global model state for the next round. This way we fine-tune the wanted LLM keeping its backbone unchanged, while the number of parameters that must be communicated between clients and server is greatly reduced.

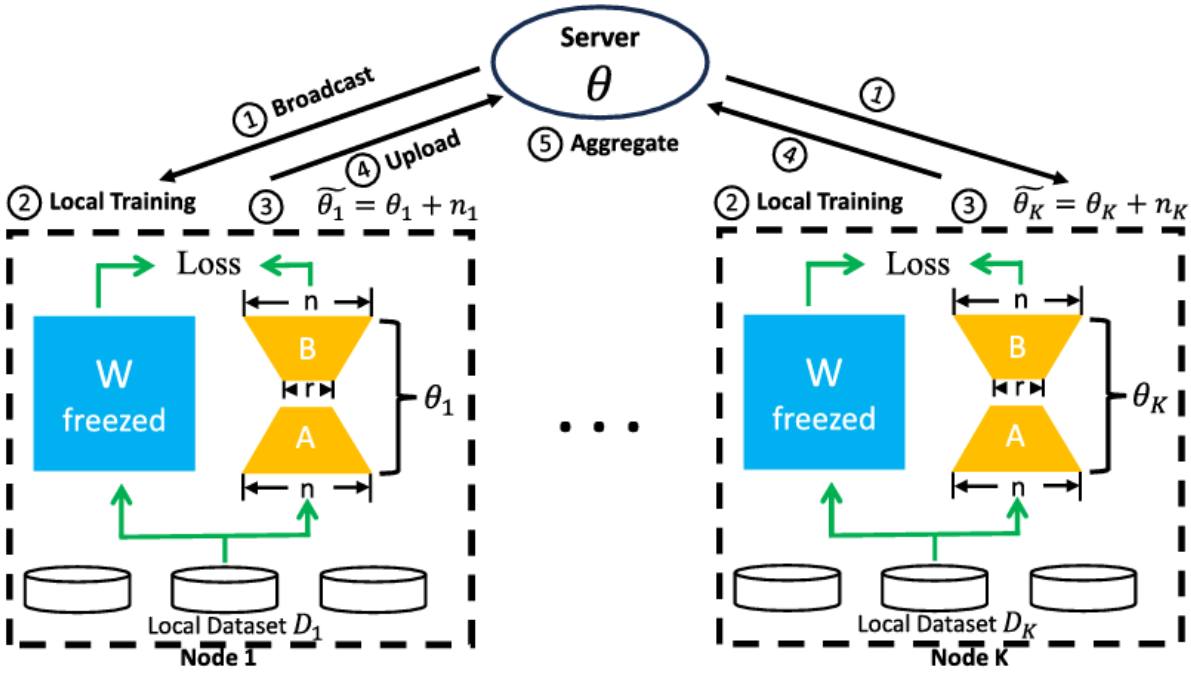

Figure 4.1: Workflow of DP-LoRA in a federated round. Each client keeps the LLM backbone frozen and fine-tunes only the low-rank adapters $(\mathbf{A}, \mathbf{B})$. The server aggregates the uploaded adapters by a weighted average to form the new global state.

Algorithm 3 DP-LoRA

1: **Input:** number of nodes K, datasets $\{D_k\}_{k=1}^K$, noise scale σ, clipping norm C, iterations T, learning rate γ, batch size B, network parameter $\boldsymbol{\theta}$
2: Initialize $\boldsymbol{\theta}$
3: **for** $t = 0$ to T **do**
4: Send $\boldsymbol{\theta}$ to all K nodes independently
5: $(\mathbf{A}_k, \mathbf{B}_k) \leftarrow \text{NODEUPDATE}(D_k, C, B, \sigma, \gamma), \quad k = 1, 2, \ldots, K$
6: $\mathbf{A} \leftarrow \sum_{k=1}^{K} \rho_k \mathbf{A}_k, \quad \mathbf{B} \leftarrow \sum_{k=1}^{K} \rho_k \mathbf{B}_k$
7: **end for**
8: **Output:** $\mathbf{A}, \mathbf{B}$
9:
10: **function** NODEUPDATE$(D_k, C, B, \sigma, \gamma)$
11: **Input**: dataset D_k, clipping norm C, learning rate γ, batch size B, noise scale σ
12: Sample $(\mathbf{x}, \mathbf{y})$ from D_k with batch size B
13: $\mathbf{g}_{\mathbf{A}_k} \leftarrow \nabla_{\mathbf{A}_k} \mathcal{L}(\mathbf{A}_k; \mathbf{x}, \mathbf{y}), \quad \mathbf{g}_{\mathbf{B}_k} \leftarrow \nabla_{\mathbf{B}_k} \mathcal{L}(\mathbf{B}_k; \mathbf{x}, \mathbf{y})$
14: $\mathbf{g}_{\mathbf{A}_k} \leftarrow \mathbf{g}_{\mathbf{A}_k} / \max(1, \|\mathbf{g}_{\mathbf{A}_k}\|_2 / C), \quad \mathbf{g}_{\mathbf{B}_k} \leftarrow \mathbf{g}_{\mathbf{B}_k} / \max(1, \|\mathbf{g}_{\mathbf{B}_k}\|_2 / C)$
15: $\mathbf{g}_{\mathbf{A}_k} \leftarrow \mathbf{g}_{\mathbf{A}_k} + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I}), \quad \mathbf{g}_{\mathbf{B}_k} \leftarrow \mathbf{g}_{\mathbf{B}_k} + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I}) \quad \triangleright\ \sigma \geq c_2 \frac{q\sqrt{T \log(1/\delta)}}{\bar{\rho}\epsilon}$
16: $\mathbf{A}_k \leftarrow \mathbf{A}_k - \gamma\, \mathbf{g}_{\mathbf{A}_k}, \quad \mathbf{B}_k \leftarrow \mathbf{B}_k - \gamma\, \mathbf{g}_{\mathbf{B}_k}$
17: **Output:** $\mathbf{A}_k, \mathbf{B}_k$
18: **end function**

At each round the server broadcasts the current model parameters (Line 4) to all clients. Each client then performs local training (Lines 10-18), keeping the backbone $\mathbf{W}$ frozen and updating only the LoRA adapter weights $(\mathbf{A}_k, \mathbf{B}_k)$. The local update procedure follows the DP-SGD algorithm [1]. First, the client computes the gradients of the loss function $\mathcal{L}$ with respect to its adapters, obtaining $\mathbf{g}_{\mathbf{A}_k}$ and $\mathbf{g}_{\mathbf{B}_k}$. These gradients are clipped if their ℓ_2 norm exceeds the threshold C, and Gaussian noise with variance proportional to C^2 is added to ensure differential privacy. The noised gradients, denoted $\tilde{\mathbf{g}}_{\mathbf{A}_k}$ and $\tilde{\mathbf{g}}_{\mathbf{B}_k}$, are then used to update the local adapters via gradient descent ($\mathbf{A}_k \leftarrow \mathbf{A}_k - \gamma\tilde{\mathbf{g}}_{\mathbf{A}_k}$, $\mathbf{B}_k \leftarrow \mathbf{B}_k - \gamma\tilde{\mathbf{g}}_{\mathbf{B}_k}$). The updated adapters are returned to the server (Line 5), which aggregates them across all clients by a weighted average (Line 6) with weights $\rho_k = N_k/N$, producing the new global adapters $(\mathbf{A}, \mathbf{B})$ for the next communication round.

While our experiments deliberately omit differential privacy, the initial method builds its guarantees on the well-studied DP-SGD framework [1]. In this setting, each client clips per-example gradients to a fixed norm C and adds Gaussian noise before the update. The analysis shows that, when this is repeated for T iterations with sampling probability $q = B/N$, the overall training procedure satisfies (ϵ, δ)-differential privacy provided the

noise scale σ is chosen appropriately. In the federated case, the same local clipping and noise are applied on every client. This ensures that the final model protects the contribution of any individual training example with the new bound incorporating $\bar{\rho} = \sqrt{\sum_k \rho_k^2}$ to reflect heterogeneous client sizes (Theorem 4 in [40]).

4.2.3 Promise of the Approach

The initial study evaluates the DP-LoRA method across multiple domains in order to test both utility and efficiency under differential privacy. Three representative application areas are considered:

- general-purpose tasks,
- financial domain applications, and
- medical domain applications.

To ensure broad coverage experiments span both moderate-scale models (GPT-2 [47] and BERT [19]) and more recent large language models such as ChatGLM-6B [69] and Llama2-7B [60]. This variety demonstrates the applicability of the method to different architectures and parameter counts.

The reported results highlight the expected privacy-utility trade-off that follows the DP-SGD-based training, stronger privacy (smaller ϵ) consistently leads to larger accuracy drops across domains and models. At the same time, the use of LoRA provides substantial communication savings compared to full fine-tuning since only the low-rank adapter weights are transmitted between clients and server.

The method demonstrated strong potential in the original study, which motivated us to omit the differential privacy component in order to better focus on the upper bound of achievable performance. In our experiments, we compare our model selection and design decisions directly against those of the paper across all reported model families.

4.3 LoRA Tuning for LLMs in a Federated Learning Setting

The aforementioned motivation and findings inspired our work. In what follows, we reformulate the proposed algorithm for federated LoRA training, present our design decisions, and highlight the changes introduced in our study. As hinted in the previous sections we keep the central idea intact, the adaptation of large language models in a federated environment through low-rank adaptation (LoRA). Our modification is to remove the constraint

of differential privacy in order to examine the full potential of the method. This perspective is important in its own right, since many real world applications of federated learning do not require strict privacy guarantees, yet still demand efficient ways of adapting large models across distributed datasets. By focusing on the non-private variant, we are able to isolate the algorithm's core strengths, parameter efficiency and collaborative training, without the accuracy trade-offs introduced by privacy noise.

4.3.1 Detailed Federated LoRA Algoirthm

To formalize our setting, we provide a refined breakdown of the federated LoRA procedure, separating the responsibilities of the server and the clients. Our formulation builds on the DP-LoRA framework [40] but emphasizes different aspects, such as the placement of adapters within transformer layers and the distinction between privacy-specific steps and the core federated mechanism.

Applying LoRA in Transformer Layers

In each transformer block we adapt the attention projections $p \in \{Q, K, V\}$. For layer l, with projection weight $\mathbf{W}_p^{(l)} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, we introduce a low-rank increment

$$\Delta \mathbf{W}_p^{(l)} \;=\; \mathbf{B}_p^{(l)}\, \mathbf{A}_p^{(l)},$$

$$\mathbf{B}_p^{(l)} \in \mathbb{R}^{d_{\text{out}} \times r}, \;\; \mathbf{A}_p^{(l)} \in \mathbb{R}^{r \times d_{\text{in}}}, \;\; r \ll \min(d_{\text{in}}, d_{\text{out}}).$$

The effective projection is then $\mathbf{W}_p^{(l)} + \Delta \mathbf{W}_p^{(l)}$, which reduces the number of trainable parameters per projection from $O(d_{\text{out}} d_{\text{in}})$ to $O\big(r(d_{\text{out}} + d_{\text{in}})\big)$.

Presentation of the Algorithm

We now present our version of the training algorithm. In contrast to the compact pseudocode of [40], we provide a more detailed and structured Python-like formulation that separates the roles of server and clients. This version makes explicit the operations involved in DP-SGD, which we later omit to obtain our non-private variant.

Algorithm 4 Server: Federated Loop

Require: pretrained backbone W, rank r, rounds T, batch size B, LR γ, client sizes $N_1, \ldots, N_K$, total $N = \sum_k N_k$

initialize $A_p^l \sim \mathcal{N}(0, \sigma_A^2)$, $B_p^l \leftarrow 0$ for all $l = 1, \ldots, L$, $p \in \{Q, K, V\}$
for $t = 1 \to T$ **do**
    broadcast full model $\theta = (W, \{A_p^l, B_p^l\})$
    for each client $k = 1 \to K$ in parallel **do**
        $A_k, B_k \leftarrow \textsc{ClientUpdate}(\theta,\ B,\ \gamma)$
    end for
    for $l = 1 \to L,\ p \in \{Q, K, V\}$ **do**
        $\rho_k \ = \ N_k / N$
        $A_p^l \ = \ \sum_{k=1}^{K} \rho_k \, A_{kp}^l$
        $B_p^l \ = \ \sum_{k=1}^{K} \rho_k \, B_{kp}^l$
    end for
end for
return $\{A_p^l, B_p^l\}$

For the DP version of Algorithm 4, we also require the clipping norm C and noise parameter σ.

Algorithm 5 Client: Federated DP-LoRA Update

function ClientUpdate($\theta, C, \gamma, B, \sigma$)
  load $\theta = (W, \{A_p^l, B_p^l\})$; keep W frozen
  $A_{p,\text{loc}}^l \leftarrow A_p^l, \quad B_{p,\text{loc}}^l \leftarrow B_p^l$ ▷ make local copies
  `requires_grad=True` on each $A_{p,\text{loc}}^l$ and $B_{p,\text{loc}}^l$
  zero accumulators:
  gradA_sum$[l][p] \leftarrow 0$, gradB_sum$[l][p] \leftarrow 0$ for all l, p
  sample batch $\{(x_i, y_i)\}_{i=1}^B \subset D_k$
  for $i = 1 \to B$ **do**
    $\ell_i \leftarrow$ `model.forward`(x_i)
    for $l = 1 \to L,\ p \in \{Q, K, V\}$ **do**
      $(gA, gB) \leftarrow$ `autograd.grad`$(\ell_i,\ [A_{p,\text{loc}}^l,\ B_{p,\text{loc}}^l])$
      $gA_{\text{clip}} \leftarrow gA \cdot \min\big(1,\ C/\|gA\|_2\big)$
      $gB_{\text{clip}} \leftarrow gB \cdot \min\big(1,\ C/\|gB\|_2\big)$
      gradA_sum$[l][p]$ += gA_{clip}, gradB_sum$[l][p]$ += gB_{clip}
    end for
  end for
  add noise & average:
  for $l = 1 \to L,\ p \in \{Q, K, V\}$ **do**
    $$\widetilde{g}A = \frac{\text{gradA\_sum}[l][p] + \mathcal{N}(0, \sigma^2 C^2 I)}{B},$$
    $$\widetilde{g}B = \frac{\text{gradB\_sum}[l][p] + \mathcal{N}(0, \sigma^2 C^2 I)}{B}$$
    $A_{p,\text{loc}}^l \leftarrow A_{p,\text{loc}}^l - \gamma\, \widetilde{g}A$
    $B_{p,\text{loc}}^l \leftarrow B_{p,\text{loc}}^l - \gamma\, \widetilde{g}B$
  end for
  return $\big(\{A_{p,\text{loc}}^l\}, \{B_{p,\text{loc}}^l\}\big)$
end function

For the non-private version, the server loop (Algorithm 4) remains unchanged. The modification lies in the client update, shown below.

Algorithm 6 Client: Federated LoRA Update

1: **function** ClientUpdate(θ, γ, B)
2: load $\theta = (W, \{A_p^l, B_p^l\})$; keep W frozen
3: $A_{p,\text{loc}}^l \leftarrow A_p^l, \quad B_{p,\text{loc}}^l \leftarrow B_p^l$ ▷ make local copies
4: `requires_grad=True` on each $A_{p,\text{loc}}^l$ and $B_{p,\text{loc}}^l$
5: **zero accumulators:**
6: gradA_sum$[l][p] \leftarrow 0$, gradB_sum$[l][p] \leftarrow 0$ for all l, p
7: sample batch $\{(x_i, y_i)\}_{i=1}^B \subset D_k$
8: **for** $i = 1 \to B$ **do**
9: $\ell_i \leftarrow$ `model.forward`(x_i)
10: **for** $l = 1 \to L,\ p \in \{Q, K, V\}$ **do**
11: $(gA, gB) \leftarrow$ `autograd.grad`$(\ell_i,\ [A_{p,\text{loc}}^l,\ B_{p,\text{loc}}^l])$
12: gradA_sum$[l][p]$ += gA, gradB_sum$[l][p]$ += gB
13: **end for**
14: **end for**
15: **average:**
16: **for** $l = 1 \to L,\ p \in \{Q, K, V\}$ **do**
17: $\overline{gA} = \dfrac{\text{gradA\_sum}[l][p]}{B}$,
18: $\overline{gB} = \dfrac{\text{gradB\_sum}[l][p]}{B}$
19: $A_{p,\text{loc}}^l \leftarrow A_{p,\text{loc}}^l - \gamma\, \overline{gA}$
20: $B_{p,\text{loc}}^l \leftarrow B_{p,\text{loc}}^l - \gamma\, \overline{gB}$
21: **end for**
22: **return** $\left(\{A_{p,\text{loc}}^l\}, \{B_{p,\text{loc}}^l\}\right)$
23: **end function**

The expanded, Python-like pseudocode above clarifies algorithmic decisions by making adapter placement, gradient paths, and aggregation indices explicit, thereby reducing ambiguity in the federated LoRA pipeline and improving reproducibility. At the same time, it modifies the original algorithm by removing the privacy components and replacing them with standard gradient averaging. The new workflow is:

1. **Broadcast:** the server sends the current global parameters to all clients (Algorithm 4, line 3).

2. **Local training:** each client k samples a batch of B examples from D_k and computes gradients for all layers and projections (Algorithm 6, lines 8-14).

3. **Averaging and update:** the gradients are averaged and used in a stochastic gradient descent step (Algorithm 6, lines 16-21).

4. **Collection:** the server receives the updated adapter weights from all clients (Algorithm 4, lines 4-6).

5. **Aggregation:** the server combines the updates by weighted average,

$$\mathbf{A}_p^l = \sum_{k=1}^{K} \rho_k \mathbf{A}_{p,k}^l, \quad \mathbf{B}_p^l = \sum_{k=1}^{K} \rho_k \mathbf{B}_{p,k}^l, \quad \rho_k = N_k/N,$$

applied across all transformer layers and projections (Algorithm 4, lines 7-11).

To make this workflow clearer, we include a figure highlighting the modified steps in the client update. While the diagram resembles the baseline workflow, it illustrates a distinct process aimed at evaluating federated LoRA in the absence of differential privacy.

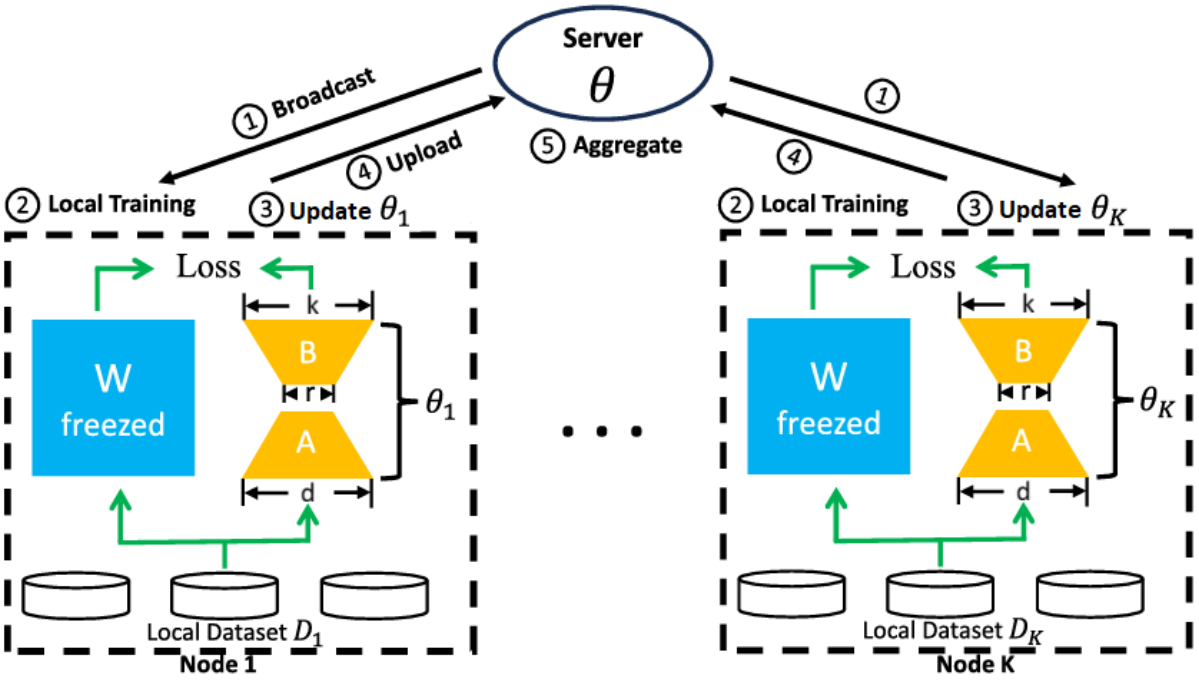

Figure 4.2: Workflow of LoRA-tuning in a federated round. Each client keeps the LLM backbone frozen and fine-tunes only the low-rank adapters $(\mathbf{A}, \mathbf{B})$. The server aggregates the uploaded adapters by a weighted average to form the new global state.

4.3.2 Choice of Model: Gemma 3

A central design decision in our work is the choice of **Gemma3-4B** as the backbone model for federated LoRA tuning. This subsection justifies that choice by highlighting key aspects of the Gemma 3 architecture and comparing them to the models used in the baseline study.

Lightweight yet strong. Gemma 3 is part of the most recent generation of open-weight models by Google DeepMind [57], available in sizes ranging from 1B to 27B parameters. In our work we adopt the 4B variant which strikes a balance, small enough to be trained and fine-tuned efficiently in federated environments, while large enough to deliver strong downstream performance. Importantly, the Gemma 3-4B is an instruction-tuned model, meaning that it has undergone additional training on datasets containing instructions paired with corresponding outputs improving its ability to follow natural language prompts. This variant achieves performance competitive with much larger models, including the 27B-parameter Gemma 2 from the previous generation [57]. This demonstrates that Gemma 3-4B inherits efficiency gains from architectural and training improvements, making it particularly well-suited as a backbone in our study.

Motivating characteristics. Compared to older baselines such as GPT-2, BERT, ChatGLM-6B, and Llama2-7B, Gemma 3 benefits from several design choices that make it more efficient and capable [57]. It is trained with a more carefully filtered dataset, avoiding duplicate examples and improving overall quality, with special emphasis on aspects such as in-context attribution. This helps to reduce hallucinations and lead to more trustworthy responses. It also uses more advanced training methods, such as reinforcement learning from human feedback (RLHF), which improves its adaptability and makes it more aligned with human preferences. In addition, Gemma 3 offers a longer context window and is designed to handle a wider variety of languages and reasoning tasks, whereas GPT-2 and BERT are mainly English-centered. These characteristics are especially valuable in a federated setting, where client data may be diverse, making Gemma 3 a strong and versatile backbone for our work.

Practical feasibility. From a systems perspective, Gemma 3-4B is easier to deploy than the larger baselines used in the DP-LoRA study, such as ChatGLM-6B and Llama2-7B, which demand significantly more compute and memory resources. By contrast, Gemma 3-4B can be fine-tuned more economically, and its open release includes quantized variants that further reduce the memory footprint [57]. This is especially important in federated environments where client hardware capabilities can vary widely.

Equally important is the fact that Gemma 3 is released as an open-weight model. While this may seem obvious given the need for local adaptation, it is nevertheless worth em-

phasizing. Unlike closed models such as GPT-4 or Claude, which require sending data to external providers, Gemma can be adapted entirely within the federated network. This aligns with recent findings showing that open LLMs are preferable for private adaptations, offering stronger privacy, lower costs, and higher performance than their closed counterparts [24]. The combination of openness, economic efficiency, and deployability therefore makes Gemma 3-4B a natural choice for us.

Memorization and Privacy. A recurring concern with large language models is their tendency to reproduce near-copies of examples seen during training, a phenomenon commonly referred to as memorization. Prior studies have shown that model parameters or outputs can occasionally repeat training text, raising risks of data leakage and unintended disclosure [13, 12, 30, 9, 44]. To quantify this, researchers define a "memorization rate", which is the ratio of model generations that match its training data compared to all model generations.[1] Audits of memorization have become standard in recent technical reports [7, 18, 56, 58, 22].

The Gemma 3 report includes such an audit and finds that its models lie at the very low end of memorization compared to prior releases such as Gemma 2, PaLM, or Gemini [57]. Specifically, Gemma 3-4B shows approximate memorization rates below 0.01% and exact memorization below 0.001%. Even these numbers are likely inflated, since the evaluation method does not fully eliminate false positives. Furthermore, severity is assessed to be low, since no memorized outputs were found to contain personal information, as verified using Google's Sensitive Data Protection service. Taken together, these results indicate that Gemma 3 exhibits low memorization risk reinforcing its suitability for our federated study.

Position relative to other models. When placed alongside the models used in the DP-LoRA study, Gemma 3-4B occupies the middle ground. Models such as GPT-2 (1.5B) and BERT (110M) are considerably smaller, which makes them easier to fine-tune but less representative of the capabilities of current generation LLMs. By contrast, ChatGLM-6B and Llama2-7B deliver stronger performance but at the cost of much greater computational and memory demands, which limits their practicality in federated deployments where client infrastructure can vary widely. Gemma 3-4B sits between them, and we selected it with the expectation that its favorable design choices would make it particularly effective in a federated setting.

[1] This does not mean that a model "contains" its training data in the sense of storing exact copies. Rather, the model encodes statistical attributes of its training data, such that under certain prompts it can regenerate portions of that data.

4.3.3 Data and Evaluation Pipeline

We developed modules of code that make federated LoRA training and evaluation clearer and, in practice, more reliable. One set of modules handles data formatting, standardizing how diverse benchmark datasets are transformed into chat-style prompts and ensuring that the model sees inputs in a consistent way and that answers are not inadvertently hinted at in the input. Another set focuses on inference and evaluation, aligning outputs with the original label space of each benchmark and using robust parsing to map free-text generations back to discrete labels. Together these components reduce ambiguity, avoid missing correct answers, and make results more consistent and reproducible across tasks.

Canonical chat templating

This component standardizes task inputs across datasets by wrapping each example in a single user-assistant exchange compatible with the Gemma 3 chat interface. Three design rules guide the templates. First, no few-shot exemplars are injected, so the model's performance reflects fine-tuning rather than prompt engineering or transient pattern learning. Second, labels are canonicalized into minimal, unambiguous strings that match the task space (e.g., "True"/"False" or sentiment categories), reducing label-token variability, making correct answers easier to detect, and promoting consistency in model outputs. Third, templates avoid leaking answer structure into the user message and never hint at a preferred style, ensuring that the model must solve the task rather than infer hidden cues.

As a concrete example, in BoolQ each instance presents the passage and question as the user turn. The task is to decide whether the statement is true or false according to the passage, with subjects ranging widely from science and law to phytology and theater. During training, the assistant turn contains the ground-truth label so that the model is trained to predict it token by token under the causal language modeling loss. At evaluation time, this turn is left empty, and the model must generate its own answer:

> **User** : Here is a passage: *passage*
> Question (answer with True or False): *question*
>
> **Model** : True or False

This keeps the output space binary and unambiguous, making both training and evaluation straightforward.

As a different type of example, WinoGrande instances present a sentence with a blank and two candidate fillers. The user turn includes the sentence and the options, while the assistant turn can be reduced to a single digit (1 or 2) that identifies the correct filler, although we also recognize the actual word to be filled as a valid answer:

User : Sentence with a blank: *sentence*
Options:
1) *option1*
2) *option2*
Fill the blank with the better option.

Model : 1 or 2

This structure keeps the answer space compact, prevents the model from copying long option text verbatim, and avoids ambiguities like "option 1" versus "1", although in practice we also implement robust parsing code that accepts either form when scoring predictions.

Other benchmarks follow the same principles. For PIQA, the model is presented with a goal and two candidate solutions, and must identify which one better achieves the goal. For TFNS, the input is a raw tweet or headline about economic or financial topics, followed by a fixed polarity question, and the model must infer the sentiment behind it. For FPB (Financial PhraseBank), the task is similar but the data comes from financial articles, where each statement is labeled for sentiment. Together these two datasets cover both the informal (social-media/newswire) and the formal (press and articles) sides of financial language.

The case of MedQuAD required more substantial preprocessing. The dataset is drawn from medical questions-answer sources and covers 37 question types (e.g., treatment, diagnosis, side effects) linked to diseases, drugs, and tests. Many answers in the raw XML files begin by repeating the user's question (e.g., Q: "What are the symptoms of X?" A: "What are the symptoms of X? The symptoms of X are..."). This duplication introduces a misleading signal where the model may gain credit by merely echoing the question rather than producing a genuine answer, or conversely be penalized when it avoids repetition and outputs a semantically valid response that differs in surface form. In both cases the overlap distorts the evaluation and makes performance estimates unreliable. To address this, we implemented a sanitization step that detects and removes leading interrogative fragments whenever they mirror the original question and occur within the first few hundred characters. This cleaning ensures that the model cannot rely on superficial repetition and must instead learn to generate the medically relevant content. While seemingly minor, this intervention is important for fair evaluation, and it may also contribute to greater training stability, highlighting how preprocessing choices can directly affect both the model and the benchmark.

Inference and robust parsing

The second set of modules ensures that evaluation is consistent and makes free-form generation compatible with discrete metrics. All prompts are reformulated in the Gemma 3 chat template, and decoding is run without sampling (using `do_sample=False`). Token generation is capped at the smallest safe limit per task, keeping evaluation lightweight while avoiding irrelevant trailing text.

Because models often answer flexibly (e.g., "Option 1," "Answer 1," or restating the choice in words), outputs are normalized with compact regular expressions that accept natural paraphrases but map them back to the canonical label space. For instance, in PIQA we match a digit with tolerant prefixes ("solution 0", "answer1"), falling back to option-text containment if necessary. In BoolQ, we accept a small lexicon of positive and negative forms, answers such as "true," "yes," or "yeah" are mapped to the label *True*, while variants like "false" or "no" are mapped to *False*. This prevents correct answers from being overlooked simply because of harmless verbosity, while avoiding task-specific hacks.

From a practical standpoint, we keep evaluation lightweight. The fine-tuned Gemma 3 model is loaded in 4-bit quantization and the testing split is formatted with the same chat template used during training. This ensures consistency between training and evaluation, keeps memory requirements low, and makes the evaluation process efficient.

Together, the formatting and inference modules ensure that evaluation reflects the model's capabilities, rather than being influenced by incidental factors that could interfere with assessing the effectiveness of the method.

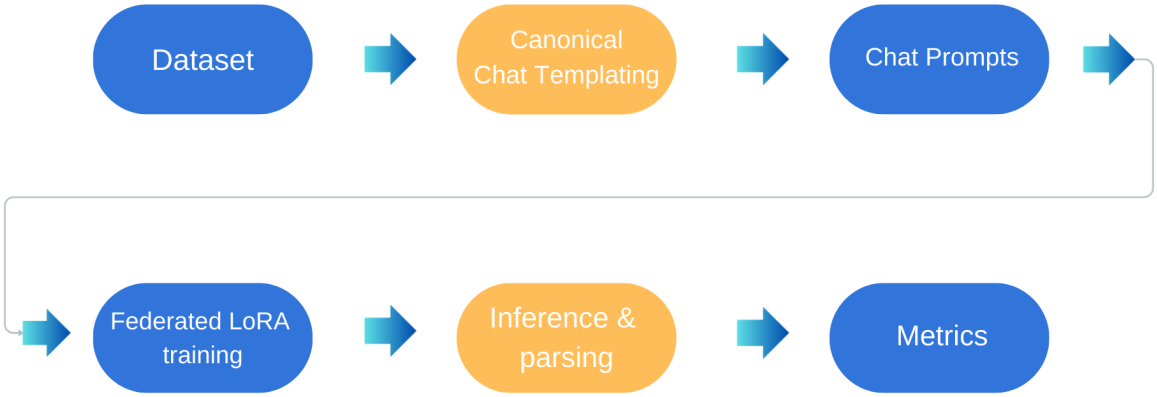

Figure 4.3: Workflow of our supporting modules. Raw datasets are first passed through canonical chat templating to produce standardized prompts, which are then used in federated LoRA training. At evaluation time, outputs are processed by inference and parsing routines to yield consistent metrics.

4.4 Experiments

We evaluate our approach across a range of benchmark datasets. To cover general-purpose reasoning and comprehension, we selected **BoolQ**, **WinoGrande**, and **PIQA**. BoolQ is a binary question-answering benchmark where the task is to decide if a passage supports a given statement. WinoGrande measures commonsense reasoning by requiring the model to fill in a blank in a sentence with the correct word from two options. PIQA (Physical Interaction Question Answering) asks the model to choose which of two solutions best accomplishes a physical goal. These benchmarks collectively test reasoning, reading comprehension and commonsense capabilities.

To extend the evaluation into the financial domain, we included the **TFNS** dataset, which consists of tweets and headlines labeled for sentiment polarity (negative, neutral, or positive). We also included **FPB** the (Financial PhraseBank) dataset, which contains statements from financial articles labeled for sentiment. These two datasets collectively represent both informal (social media) and formal (news and reports) financial language.

We should also note that all the aforementioned datasets were also used in the original

study, while the ones we left out were either unavailable or at least could not be located. In the case of MedQuAD, the dataset was accessible (although altered due to copyright restrictions) but required considerably more computational resources to be executed at least with a rank of $r = 128$ and a batch size of 6 in order to remain close to the original experimental setup.

Furthermore, as previously discussed, the baseline DP-LoRA study [40] evaluates performance across four models: GPT-2 (1.5B parameters), BERT (110M), ChatGLM-6B, and LLaMA2-7B. Two of these models (GPT-2 and BERT) are substantially smaller than our Gemma3-4B backbone, while ChatGLM-6B is moderately larger and LLaMA2-7B is roughly twice its size. We compare our method against all of them using the reported results from the corresponding tables as reference.

4.4.1 Training Constraints

Due to hardware limitations, most of our experiments were conducted with a rank of $r = 128$ and only 4-5 federated rounds, compared to the $r = 512$ and 50 rounds used in the baseline study. In practice, this means that our models were trained under more constrained and less favorable conditions. The reduced rank partially compensates for the smaller number of rounds, since fewer parameters are trained per round, but overall our setup involves significantly fewer optimization steps.

For certain datasets, additional adjustments were necessary. For example, PIQA was trained with a batch size of 6 instead of 8 to fit within GPU memory constraints. Similarly while training with $r = 256$ was feasible for some datasets, we opted to keep $r = 128$ to ensure more stable training given the limited number of rounds. Importantly, at no point did we alter hyperparameters in ways that would grant an advantage over the reported baselines. On the contrary, across all experiments our configuration involved either fewer trainable parameters, fewer training rounds or smaller batch sizes.

Therefore our results should be viewed as conservative, since they were achieved under less favorable training conditions than the baseline study. Whenever our Gemma 3 model matches or exceeds the reported baselines, it does so despite operating under stricter constraints. We emphasize this to show that the observed performance gains cannot be attributed to more favorable experimental conditions, but rather to the effectiveness of the federated LoRA strategy and the robustness of Gemma 3 as a foundation model. These results thus validate the methodological modifications introduced in our approach.

4.4.2 Results

In this subsection, we provide the execution details and results obtained across the evaluated datasets. We begin with the three general-domain benchmarks, followed by the two financial-domain datasets. For clarity, we highlight the best overall result in bold and underline the best result reported in the original study. This format allows for an intuitive comparison between our approach and the baseline models.

BoolQ

Rank (r)	Ours (Gemma3-4B)	LLaMA2-7B	ChatGLM-6B	BERT	GPT-2
512	–	<u>76.4</u>	61.5	45.8	61.9
128	**88.04**	50.3	51.3	–	–

Table 4.1: BoolQ dataset. Performance of our approach compared to all models reported in [40]. GPT-2 results are from Table 16, BERT from Table 17, ChatGLM-6B from Table 12, LLaMA2-7B from Table 5, and the $r = 128$ results from Table 5.

For BoolQ, our model was trained with $r = 128$, batch size 8 and 4 federated rounds. Under these settings, our Gemma3-4B fine-tuned model achieved an accuracy of 88.04% compared to the best reported LLaMA2-7B baseline of 76.3%. This corresponds to an absolute gain of 11.7 percentage points or a relative improvement of 15.4%.

WinoGrande

Rank (r)	Ours (Gemma3-4B)	LLaMA2-7B	ChatGLM-6B	BERT	GPT-2
512	–	<u>70.0</u>	63.8	51.3	48.3
128	**77.35**	69.7	44.1	–	–

Table 4.2: WinoGrande dataset. Performance of our approach compared to all models reported in [40]. GPT-2 from Table 16, BERT from Table 11, ChatGLM-6B from Table 12, LLaMA2-7B from Table 13, and the $r = 128$ results from Table 5.

For WinoGrande, we used the `winogrande-xl` variant containing 43.4k sentences, matching the one used in the original study. Our model was trained with a rank of $r = 128$, batch size 8 and 5 federated rounds. Each client processed approximately 0.4 of an epoch per round due to the large local subsets assigned to them, which increased the computational cost and prolonged training. Under these settings, our Gemma3-4B fine-tuned model achieved

an accuracy of 77.35% compared to the best reported LLaMA2-7B baseline of 70.0% . This corresponds to an absolute gain of 7.35 percentage points or a relative improvement of approximately 10.5%.

PIQA

Rank (r)	Ours (Gemma3-4B)	LLaMA2-7B	ChatGLM-6B	BERT	GPT-2
512	–	<u>80.4</u>	69.3	51.3	53.8
128	**83.79**	78.5	56.7	–	–

Table 4.3: PIQA dataset. Performance of our approach compared to all models reported in [40]. GPT-2 from Table 16, BERT from Table 17, ChatGLM-6B from Table 12, LLaMA2-7B from Table 5, and the $r = 128$ results from Table 5.

For PIQA, training was conducted with a rank of $r = 128$, batch size 6 and 5 federated rounds. Since each client handled a comparatively large local subset, the total training covered roughly 0.6 of an epoch per client. Despite the reduced number of rounds, our Gemma3-4B fine-tuned model reached an accuracy of 83.79%, outperforming the best reported LLaMA2-7B baseline of 80.4%. This represents an absolute improvement of 3.39 percentage points and a relative gain of approximately 4.2%.

Both WinoGrande and PIQA posed particular challenges given our limited training regime. Because of their relatively large sizes the number of local steps each client could complete per round was restricted, about 0.4 of an epoch for WinoGrande and 0.6 for PIQA, leaving fewer effective passes over the data. This naturally constrained the model's ability to further improve its accuracy, especially for the larger WinoGrande dataset. While WinoGrande is demanding due to its scale and the fine-grained commonsense reasoning required to fill gaps in text, PIQA is inherently more difficult from a task perspective, since each instance presents two realistic physical solutions to a goal and the model must infer which one would succeed in the real world. However our model still managed to outperform the baseline results.

TFNS

Rank (r)	Ours (Gemma3-4B)	LLaMA2-7B	ChatGLM-6B	BERT	GPT-2
512	–	69.6	<u>70.1</u>	57.3	66.0
128	**89.57**	–	–	–	–

Table 4.4: TFNS dataset. Performance of our approach compared to all models reported in [40]. GPT-2 from Table 10, BERT from Table 11, ChatGLM-6B from Table 12, LLaMA2-7B from Table 13.

For TFNS, our model was trained with a rank of $r = 128$, batch size 8 and 4 federated rounds. Under these conditions, our Gemma3-4B fine-tuned model achieved an accuracy of 89.57% outperforming the best reported ChatGLM-6B baseline of 70.1%. This translates to an absolute gain of 19.47 percentage points and a relative improvement of approximately 27.8%.

FPB

Rank (r)	Ours (Gemma3-4B)	LLaMA2-7B	ChatGLM-6B	BERT	GPT-2
512	–	59.1	55.5	<u>65.5</u>	59.8
128	**86.39**	–	–	–	–

Table 4.5: FPB dataset. Performance of our approach compared to all models reported in [40]. GPT-2 from Table 10, BERT from Table 11, ChatGLM-6B from Table 12, LLaMA2-7B from Table 13.

For FPB, training was conducted with a rank of $r = 128$, batch size 8, and 5 federated rounds. Under these conditions, our Gemma3-4B fine-tuned model reached an accuracy of 86.39%, surpassing the best baseline reported in [40], achieved by BERT at 65.5%. This marks an absolute improvement of 20.89 percentage points and a relative gain of roughly 31.9%.

We also note that the strong performance on the financial-domain datasets sparked our interest for a closer examination. The observed gains exceeding 20 percentage points over the best reported baselines raised the question of whether these results arose from our federated fine-tuning procedure or were influenced by the inherent capabilities of the Gemma-3 model itself. We explore this question further in the next subsection.

4.4.3 Overall Performance Comparison

Table 4.6 summarizes the results across all evaluated datasets.

Dataset	Rank (r)	Ours (Gemma3-4B)	LLaMA2-7B	ChatGLM-6B	BERT	GPT-2
BoolQ	512	–	<u>76.3</u>	61.5	45.8	61.9
	128	**88.04**	50.3	51.3	–	–
WinoGrande	512	–	<u>70.0</u>	63.8	51.3	48.3
	128	**77.35**	69.7	44.1	–	–
PIQA	512	–	<u>80.4</u>	69.3	51.3	53.8
	128	**83.79**	78.5	56.7	–	–
TFNS	512	–	69.6	<u>70.1</u>	57.3	66.0
	128	**89.57**	–	–	–	–
FPB	512	–	59.1	55.5	<u>65.5</u>	59.8
	128	**86.39**	–	–	–	–

Table 4.6: Overall comparison across all evaluated datasets. Bold numbers indicate the best result for each dataset, while underlined values denote the strongest non-private baselines reported in the original study [40]. For detailed references to the specific tables from which each baseline result was taken, see the corresponding per-dataset tables provided earlier in this section.

From the collective table above, it is clear that the strongest results among their models generally come from the largest one, LLaMA2-7B, while the smaller models (specifically BERT) managed to outperform the larger ones only in a single benchmark (FPB). This pattern aligns with the expected relationship between model size and performance, as larger architectures typically possess greater capacity for generalization. Nevertheless, our fine-tuned Gemma3-4B consistently achieved higher scores across all datasets, despite being trained under more constrained conditions and with substantially fewer parameters. This demonstrates that high performance can be achieved without relying on large parameter counts. By pairing an appropriately chosen backbone with our carefully structured federated LoRA adaptation, our approach attains competitive accuracy under constrained conditions reinforcing the soundness of the design choices made throughout this work.

Interestingly, from Table 4.6, we can observe that our model achieved particularly strong results on the financial-domain datasets. In both TFNS and FPB, the fine-tuned Gemma3-4B outperformed the best models reported in [40] by relative improvements exceeding 25%.

These substantial gains motivated a closer examination of whether the improvements stem from the federated fine-tuning procedure itself rather than from any inherent strength of the base model.

Model	TFNS	FPB
Gemma 3 (no tuning)	43.38	44.54
Gemma 3 tuned	89.57	86.39

Table 4.7: Performance of the Gemma-3 model on financial-domain datasets before and after applying federated LoRA fine-tuning.

To test this, we evaluated the untuned Gemma-3 model on the same financial datasets and compared its performance to our fine-tuned version. As shown in Table 4.7, the base model performs close to random guessing on both datasets, achieving around 43-44% accuracy. After applying our federated LoRA fine-tuning accuracy nearly doubles, reaching 89.57% on TFNS and 86.39% on FPB. This demonstrates that the observed improvements stem directly from our adaptation procedure rather than from any inherent advantage of the base model, and highlights the importance of targeted fine-tuning even for modern instruction-tuned backbones.

Chapter 5

Conclusions

This final chapter summarizes our study. We revisit the main objectives set at the beginning of the thesis, outline how they were achieved, and highlight the significance of the results obtained through our experiments. Finally, we discuss possible extensions and future research directions that could build upon this work.

5.1 Result Summary

This thesis presented a non-private variant of the DP-LoRA framework, designed to study the effectiveness of federated fine-tuning for large language models in the absence of differential privacy. By removing the privacy component, the intention was to isolate and evaluate the core capabilities of the federated LoRA algorithm, measuring its true potential in collaborative adaptation scenarios without the additional performance degradation introduced by privacy noise.

To support this goal, we implemented two complementary modules that ensured consistency and reproducibility across all stages of training and evaluation. The first, a data formatting module, was responsible for constructing clear and leakage-free prompts for each dataset, converting examples into a standardized chat-style format suitable for large language models. The second, an inference and parsing component, enabled robust and deterministic evaluation across tasks. Together, these tools were developed to minimize external variability and ensure that the reported outcomes reflect the genuine effectiveness of the model-algorithm pair.

Our experiments employed the Gemma3-4B open-weight model as the backbone, selected for its promising performance reported in the Gemma technical report and its relatively lightweight architecture, which makes it suitable for federated learning environments where client hardware resources can vary considerably. Despite being smaller than the largest models used in the baseline DP-LoRA study (LLaMA2-7B and ChatGLM-6B) and

trained under significantly constrained conditions, lower rank ($r = 128$), fewer communication rounds and in some cases smaller batch sizes, our approach consistently surpassed all reported baselines. Across general-domain benchmarks such as BoolQ, WinoGrande and PIQA, as well as financial-domain datasets like TFNS and FPB, our model achieved superior accuracy. The improvements were particularly striking in the financial domain, where performance gains exceeded 20 percentage points over the strongest reported baselines.

Overall, this study validates the methodological refinements introduced in our work and highlights the potential of federated LoRA as a practical approach for distributed model training. Furthermore, although it was not our initial goal, the findings demonstrate that even under resource-limited settings, high performance can be achieved through careful model selection and efficient federated fine-tuning.

5.2 Future Work

This work was conducted under constrained computational resources, which limited the number of training rounds and the parameter dimensions used. A natural continuation would therefore be to rerun the experiments under the same conditions as the original study, using larger ranks and more communication rounds, to allow for a fully parallel comparison of performance trends. While expanding the evaluation to additional datasets, such as MedQuAD, would also provide a broader assessment of the model's adaptability across domains.

A further research direction would be to reintroduce differential privacy into the training process, enabling a comparison between noth our non-private results and the differentially private outcomes reported in the original DP-LoRA study. This would help quantify the exact trade-off between privacy preservation and utility in realistic federated environments. Additional experiments could also involve hyperparameter optimization across learning rates, LoRA ranks and the number of communication rounds, to identify potential overfitting in model performance.

Another promising line of work involves extending this study across the Gemma-3 model family. For instance, experimenting with the 1B, 12B and 27B variants [1] could help establish a practical mapping between task complexity, client resources and the appropriate model size. Such an analysis would provide insights into when it becomes worthwhile to move to a larger model, ensuring that computational resources are not wasted on unnecessarily heavy architectures when smaller ones are sufficient. A similar investigation could also be combined with alternative adaptation strategies, offering a more systematic understanding of parameter-efficient fine-tuning under federation.

Finally, it would be particularly interesting to extend the current setup to non-IID sce-

[1] After the initial release, a 270M-parameter variant of Gemma-3 was also released.

narios, where client data distributions differ and more specifically to cases of imbalanced clients, where certain nodes contribute substantially more data than others. Such experiments would offer valuable insights into the robustness and fairness of federated LoRA under varying real-world conditions.

Bibliography

[1] 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*. 2016, pp. 308–318.

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. "Gpt-4 technical report". In: *arXiv preprint arXiv:2303.08774* (2023).

[3] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. "Intrinsic dimensionality explains the effectiveness of language model fine-tuning". In: *arXiv preprint arXiv:2012.13255* (2020).

[4] Jay Alammar. *The Illustrated Transformer*. 2018.

[5] Q Jiang Albert, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, and Devendra Singh Chaplot. "Mistral 7B". In: *arXiv preprint* (2023).

[6] Galen Andrew, Om Thakkar, Brendan McMahan, and Swaroop Ramaswamy. "Differentially private learning with adaptive clipping". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 17455–17466.

[7] Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. "Palm 2 technical report". In: *arXiv preprint arXiv:2305.10403* (2023).

[8] Anthropic. *Introducing Claude*. Anthropic Website. URL https://www.anthropic.com/index/introducing-claude. Mar. 2023.

[9] Stella Biderman, Usvsn Prashanth, Lintang Sutawika, Hailey Schoelkopf, Quentin Anthony, Shivanshu Purohit, and Edward Raff. "Emergent and predictable memorization in large language models". In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 28072–28090.

[10] Keith Bonawitz, Fariborz Salehi, Jakub Konečnỳ, Brendan McMahan, and Marco Gruteser. "Federated learning with autotuned communication-efficient secure aggregation". In: *2019 53rd Asilomar Conference on Signals, Systems, and Computers*. IEEE. 2019, pp. 1222–1226.

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

[12] Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. "Quantifying memorization across neural language models". In: *The Eleventh International Conference on Learning Representations*. 2022.

[13] Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. "Extracting training data from large language models". In: *30th USENIX security symposium (USENIX Security 21)*. 2021, pp. 2633–2650.

[14] Nicolas Carlini, Jamie Hayes, Milad Nasr, Matthew Jagielski, Vikash Sehwag, Florian Tramer, Borja Balle, Daphne Ippolito, and Eric Wallace. "Extracting training data from diffusion models". In: *32nd USENIX security symposium (USENIX Security 23)*. 2023, pp. 5253–5270.

[15] Zachary Charles and Jakub Konečnỳ. "On the outsized importance of learning rates in local update methods". In: *arXiv preprint arXiv:2007.00878* (2020).

[16] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. *Vicuna: An Open-Source bot Impressing GPT-4 with 90%* ChatGPT Quality*. Mar. 2023. URL: https://lmsys.org/blog/2023-03-30-vicuna/.

[17] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael Jordan, Joseph E Gonzalez, Ion Stoica, et al. "Chatbot arena: An open platform for evaluating llms by human preference". In: *Forty-first International Conference on Machine Learning*. 2024.

[18] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. "Palm: Scaling language modeling with pathways". In: *Journal of Machine Learning Research* 24.240 (2023), pp. 1–113.

[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 2019, pp. 4171–4186.

[20] Haonan Duan, Adam Dziedzic, Nicolas Papernot, and Franziska Boenisch. "Flocks of stochastic parrots: Differentially private prompt learning for large language models". In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 76852–76871.

[21] Haonan Duan, Adam Dziedzic, Mohammad Yaghini, Nicolas Papernot, and Franziska Boenisch. "On the Privacy Risk of In-context Learning". In: *The 61st Annual Meeting Of The Association For Computational Linguistics*. 2023.

[22] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. "The llama 3 herd of models". In: *arXiv e-prints* (2024), arXiv–2407.

[23] Hubert Eichner, Tomer Koren, Brendan McMahan, Nathan Srebro, and Kunal Talwar. "Semi-cyclic stochastic gradient descent". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 1764–1773.

[24] Vincent Hanke, Tom Blanchard, Franziska Boenisch, Iyiola E Olatunji, Michael Backes, and Adam Dziedzic. "Open LLMs are necessary for current private adaptations and outperform their closed alternatives". In: *Proceedings of the 38th International Conference on Neural Information Processing Systems*. 2024, pp. 1220–1250.

[25] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. "Training compute-optimal large language models". In: *arXiv preprint arXiv:2203.15556* (2022).

[26] Junyuan Hong, Jiachen Wang, Chenhui Zhang, Zhangheng Li, Bo Li, and Zhangyang Wang. "Dp-opt: Make large language model your privacy-preserving prompt engineer". In: International Conference on Learning Representations (ICLR) 2024. 2024.

[27] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. "Parameter-efficient transfer learning for NLP". In: *International conference on machine learning*. PMLR. 2019, pp. 2790–2799.

[28] Jeremy Howard and Sebastian Ruder. "Universal language model fine-tuning for text classification". In: *arXiv preprint arXiv:1801.06146* (2018).

[29] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. "Lora: Low-rank adaptation of large language models." In: *ICLR* 1.2 (2022), p. 3.

[30] Daphne Ippolito, Florian Tramèr, Milad Nasr, Chiyuan Zhang, Matthew Jagielski, Katherine Lee, Christopher A Choquette-Choo, and Nicholas Carlini. "Preventing verbatim memorization in language models gives a false sense of privacy". In: *arXiv preprint arXiv:2210.17546* (2022).

[31] 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". In: *Foundations and trends® in machine learning* 14.1–2 (2021), pp. 1–210.

[32] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. "Scaling laws for neural language models". In: *arXiv preprint arXiv:2001.08361* (2020).

[33] Brian Lester, Rami Al-Rfou, and Noah Constant. "The Power of Scale for Parameter-Efficient Prompt Tuning". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 3045–3059. URL: https://aclanthology.org/2021.emnlp-main.243/.

[34] Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. "Measuring the intrinsic dimension of objective landscapes". In: *arXiv preprint arXiv:1804.08838* (2018).

[35] Xiang Lisa Li and Percy Liang. "Prefix-tuning: Optimizing continuous prompts for generation". In: *arXiv preprint arXiv:2101.00190* (2021).

[36] Zhaojiang Lin, Andrea Madotto, and Pascale Fung. "Exploring versatile generative language model via parameter-efficient transfer learning". In: *arXiv preprint arXiv:2004.03829* (2020).

[37] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. "P-Tuning: Prompt Tuning Can Be Comparable to Fine-tuning Across Scales and Tasks". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 61–68. URL: https://aclanthology.org/2022.acl-short.8/.

[38] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. "P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks". In: *arXiv preprint arXiv:2110.07602* (2021).

[39] Xiao-Yang Liu, Guoxuan Wang, Hongyang Yang, and Daochen Zha. "Fingpt: Democratizing internet-scale data for financial large language models". In: *arXiv preprint arXiv:2307.10485* (2023).

[40] Xiao-Yang Liu, Rongyi Zhu, Daochen Zha, Jiechao Gao, Shan Zhong, Matt White, and Meikang Qiu. "Differentially private low-rank adaptation of large language model using federated learning". In: *ACM Transactions on Management Information Systems* 16.2 (2025), pp. 1–24.

[41] Renqian Luo, Liai Sun, Yingce Xia, Tao Qin, Sheng Zhang, Hoifung Poon, and Tie-Yan Liu. "BioGPT: generative pre-trained transformer for biomedical text generation and mining". In: *Briefings in bioinformatics* 23.6 (2022), bbac409.

[42] Wang Luping, Wang Wei, and Li Bo. "CMFL: Mitigating communication overhead for federated learning". In: *2019 IEEE 39th international conference on distributed computing systems (ICDCS)*. IEEE. 2019, pp. 954–964.

[43] 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*. PMLR. 2017, pp. 1273–1282.

[44] Milad Nasr, Nicholas Carlini, Jonathan Hayase, Matthew Jagielski, A Feder Cooper, Daphne Ippolito, Christopher A Choquette-Choo, Eric Wallace, Florian Tramèr, and Katherine Lee. "Scalable extraction of training data from (production) language models". In: *arXiv preprint arXiv:2311.17035* (2023).

[45] OpenAI. *https://openai.com*. URL https://openai.com/.

[46] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. "Adapterfusion: Non-destructive task composition for transfer learning". In: *arXiv preprint arXiv:2005.00247* (2020).

[47] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. "Language models are unsupervised multitask learners". In: *OpenAI blog* 1.8 (2019), p. 9.

[48] Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. "Scaling language models: Methods, analysis & insights from training gopher". In: *arXiv preprint arXiv:2112.11446* (2021).

[49] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. "Exploring the limits of transfer learning with a unified text-to-text transformer". In: *Journal of machine learning research* 21.140 (2020), pp. 1–67.

[50] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. "Learning multiple visual domains with residual adapters". In: *Advances in neural information processing systems* 30 (2017).

[51] Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. "Adapterdrop: On the efficiency of adapters in transformers". In: *arXiv preprint arXiv:2010.11918* (2020).

[52] Karan Singhal, Shekoofeh Azizi, Tao Tu, S Sara Mahdavi, Jason Wei, Hyung Won Chung, Nathan Scales, Ajay Tanwani, Heather Cole-Lewis, Stephen Pfohl, et al. "Large language models encode clinical knowledge". In: *Nature* 620.7972 (2023), pp. 172–180.

[53] Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, et al. "Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model". In: *arXiv preprint arXiv:2201.11990* (2022).

[54] Xinyu Tang, Richard Shin, Huseyin A Inan, Andre Manoel, Fatemehsadat Mireshghallah, Zinan Lin, Sivakanth Gopi, Janardhan Kulkarni, and Robert Sim. "Privacy-Preserving In-Context Learning with Differentially Private Few-Shot Generation". In: *The Twelfth International Conference on Learning Representations (ICLR) 2024*. 2024.

[55] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. "Gemini: a family of highly capable multimodal models". In: *arXiv preprint arXiv:2312.11805* (2023).

[56] Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context". In: *arXiv preprint arXiv:2403.05530* (2024).

[57] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. "Gemma 3 technical report". In: *arXiv preprint arXiv:2503.19786* (2025).

[58] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. "Gemma 2: Improving open language models at a practical size". In: *arXiv preprint arXiv:2408.00118* (2024b).

[59] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. "Llama: Open and efficient foundation language models". In: *arXiv preprint arXiv:2302.13971* (2023).

[60] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307.09288 [cs.CL]. URL: https://arxiv.org/abs/2307.09288.

[61] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[62] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H Vincent Poor. "Tackling the objective inconsistency problem in heterogeneous federated optimization". In: *Advances in neural information processing systems* 33 (2020), pp. 7611–7623.

[63] BigScience Workshop et al. *BLOOM: A 176B-Parameter Open-Access Multilingual Language Model*. 2023. arXiv: 2211.05100 [cs.CL]. URL: https://arxiv.org/abs/2211.05100.

[64] Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. "Bloomberggpt: A large language model for finance". In: *arXiv preprint arXiv:2303.17564* (2023).

[65] Tong Wu, Ashwinee Panda, Jiachen T Wang, and Prateek Mittal. "Privacy-preserving in-context learning for large language models". In: *12th International Conference on Learning Representations, ICLR 2024*. 2024.

[66] Jinjin Xu, Wenli Du, Yaochu Jin, Wangli He, and Ran Cheng. "Ternary compression for communication-efficient federated learning". In: *IEEE Transactions on Neural Networks and Learning Systems* 33.3 (2020), pp. 1162–1176.

[67] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. "Federated machine learning: Concept and applications". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 10.2 (2019), pp. 1–19.

[68] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. "Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models". In: *arXiv preprint arXiv:2106.10199* (2021).

[69] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. "Glm-130b: An open bilingual pre-trained model". In: *arXiv preprint arXiv:2210.02414* (2022).

[70] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. "A survey of large language models". In: *arXiv preprint arXiv:2303.18223* 1.2 (2023).