Federated Learning with Non-IID Data: Techniques, Challenges, and Implementation

Kamal Nehra (2022MT61979), Ryan Azim Shaikh (2022MT11923),
Keshav Rai (2022MT61968), Hemant Ramgarhia (2022MT11854)

Contents

1 Federated Learning: Historical and Technical Overview

1.1 Introduction and Motivation

Federated Learning (FL) is a distributed machine learning paradigm in which multiple clients (e.g., user devices or separate organizations) collaboratively train a model under the coordination of a central server, while *keeping all training data decentralized* on the clients [1].

In contrast to traditional centralized learning—where data is aggregated into a central server—FL keeps data at the source and brings the model to the data. The core idea, first formalized by McMahan *et al.* (2017), is for each client to compute updates (e.g., gradient steps) on its local data and send only those updates to a central server, which aggregates them to improve a global model [2].

By never transferring raw data, FL aligns with principles of privacy preservation and data minimization [1]. This framework has gained importance with the rise of edge computing (e.g., smartphones, IoT devices) and the need to utilize distributed data that is either too sensitive or voluminous to centralize.

FL is now widely applied in diverse domains—from keyboard next-word prediction on mobile devices to collaborative AI in healthcare—where data privacy, user ownership, or legal constraints play a crucial role.

In a typical FL setup (also called *cross-device FL*), a central server orchestrates model training through iterative communication rounds. In each round, a subset of clients download the current global model, locally update it by running a few epochs of Stochastic Gradient Descent (SGD), and then upload the resulting updates (e.g., weights or gradients) back to the server. The server then aggregates these updates (e.g., via weighted averaging) into a refined global model [2].

This process continues iteratively until convergence. A key challenge in FL is handling *heterogeneous* data: clients often possess data that is not independent and identically distributed (non-IID), and may vary significantly in quantity and quality [2]. Furthermore, FL systems must be resilient to client dropout, intermittent connectivity, and resource-constrained devices, making FL a fundamentally different and modern learning paradigm.

1.2 Origins in Distributed Learning and Optimization

Federated Learning (FL) did not arise in isolation; it evolved from prior research in distributed machine learning and optimization. In the decade preceding FL's inception, large-scale machine learning was typically conducted in data centers on clusters of machines, using frameworks we now call *distributed learning*. A common approach was to partition a large dataset across multiple worker nodes and train a model in parallel, synchronizing updates through a centralized parameter server or collective communication mechanisms such as All-Reduce in HPC clusters.

For instance, the Parameter Server architecture introduced by Li *et al.* (2014) [7]enabled distributed SGD by aggregating gradients from worker nodes, assuming the data could be split i.i.d. across a high-speed network. These classical distributed learning methods were primarily designed to *parallelize computation for speedup* and scale, making it feasible to handle massive datasets [1].

These systems typically assumed that a single organization owned the data and infrastructure, with reliable and always-online servers operating within a controlled environment. Furthermore, each node was assumed to hold a representative shard of the full dataset, simplifying the application of global SGD. Under such assumptions, synchronized and asynchronous SGD methods (e.g., Hogwild) could significantly accelerate deep model training.

However, by the mid-2010s, new challenges emerged as machine learning extended beyond the confines of data centers. At the same time, a growing body of literature explored *decentralized optimization* and *gossip algorithms*, which required no central server. In these methods, nodes communicate peer-to-peer in a network graph, iteratively averaging their parameters with neighbors to reach consensus [1].

These decentralized algorithms, rooted in control theory and multi-agent systems, were also explored in the context of distributed ML. Lian *et al.* (2017) demonstrated that under certain conditions, gossip-based SGD could match the convergence speed of centralized parameter-server methods—even within data centers [1]. These approaches eliminated single points of failure and balanced communication load, but relied on assumptions like a fully connected and synchronized peer network, which are infeasible at the scale of millions of mobile devices.

Fully peer-to-peer training is difficult to orchestrate at such scale, especially when managing global model versions, synchronization, and convergence without central control. As a result, decentralized optimization remained largely a theoretical or specialized approach before FL.

By 2015, researchers recognized that neither centralized data-center models nor decentralized methods were suitable for the new frontier of *edge-device learning*. Jakub Konečný and colleagues at Google introduced the concept of "Federated Optimization," defining a scenario with massive numbers of devices (e.g., smartphones), each holding a tiny, non-

representative local dataset, collaborating to train a global model without sharing raw data [3].

Their pioneering work, presented at an AI workshop in 2015, emphasized that existing distributed training algorithms were unsuitable for this scenario, and that *communication efficiency* would be of utmost importance. This work laid the groundwork for FL by identifying its key constraints: unreliable, slow devices, non-IID data, and the need for communication-efficient distributed optimization.

In essence, this was the conception of Federated Learning: distributed optimization pushed to the edge, with strong emphasis on privacy, heterogeneity, and minimal communication.

1.3 Emergence of Federated Learning

Building on foundational ideas in distributed optimization and edge computing, the first major Federated Learning (FL) algorithm and its benchmark results were introduced by McMahan *et al.* in early 2017, in the landmark paper titled *Communication-Efficient Learning of Deep Networks from Decentralized Data* [2]. This work formally coined the term *Federated Learning* and proposed the Federated Averaging (FedAvg) algorithm, which has since become a cornerstone of the field.

The FedAvg algorithm balances local computation with global communication: instead of computing a single gradient update per round (as in traditional distributed SGD), each participating client performs multiple local SGD steps on its private dataset before sending the updated model parameters to the server. The central server then aggregates these updates—typically through weighted averaging—to form the new global model for the next round.

This architecture significantly reduces communication overhead. McMahan *et al.* reported that FedAvg required $10\times$ to $100\times$ fewer communication rounds compared to synchronized SGD. This improvement is especially critical in FL scenarios, where mobile or edge devices communicate over slow and costly wireless networks.

Another key insight from the paper was that FedAvg remains effective even when data is highly non-IID and unbalanced across clients—something that was not anticipated given traditional ML theory's reliance on IID assumptions. Demonstrating that accurate models can be learned in such a challenging setting was a significant milestone.

Equally important, the paper articulated the **real-world motivations** for Federated Learning. Modern mobile devices generate rich, sensitive data (e.g., messages, photos, voice recordings) that could significantly improve AI models, but transferring this data to central servers is often impractical due to privacy concerns and bandwidth limitations. FL offers an alternative approach: *training models directly on devices while sharing only aggregated updates*, thus preserving privacy and minimizing communication.

This privacy-aware design aligns with regulatory and user expectations and fits naturally with the growing trend of moving computation to the network edge—a concept often referred to as "on-device AI." Enabled by increasingly powerful mobile hardware, FL emerged between 2016 and 2017 as a compelling framework to leverage vast, decentralized datasets that were previously untapped due to privacy and infrastructural limitations.

The initial success of FedAvg served as a strong proof-of-concept, laying the groundwork for FL as a practical and scalable paradigm for collaborative AI.

1.4 Key Developments and Milestones in FL

Following its introduction, Federated Learning (FL) quickly became a prominent area of research, leading to many theoretical advances and real-world deployments. A major milestone was the deployment of FL for Google's Gboard virtual keyboard. Hard *et al.* (2018) used FL to train a recurrent neural network for next-word prediction directly on user devices using the Federated Averaging algorithm [4]. They found that models trained using real on-device typing data outperformed centrally trained models, due to FL's ability to leverage private and diverse data without it ever leaving the devices.

This was a landmark demonstration of FL's real-world viability. It showed that training complex deep learning models on millions of mobile devices was not only feasible but also effective, providing improved accuracy while ensuring that sensitive user data remained private. Furthermore, it introduced the concept of "privacy by default," since only aggregated model updates—not raw data—were communicated to Google's servers.

Around the same time, Google began exploring FL for other applications such as speech models, and released the open-source framework *TensorFlow Federated* to support academic and industrial research in the field.

Between 2018 and 2020, numerous challenges in FL were addressed. Communication efficiency remained a key concern, leading to research in model update compression, quantization, gradient sparsification, and adaptive optimization strategies. On the privacy front, formal guarantees were proposed using techniques like *differential privacy*, ensuring that model updates could not leak individual information. Secure Aggregation protocols were also introduced to encrypt model updates during transmission. By 2019, Google had demonstrated an FL system with formal user-level differential privacy in Gboard [5].

Another challenge was device heterogeneity. In cross-device FL, clients have diverse hardware and may drop out intermittently. FedProx, introduced by Li *et al.* (2018), improved FedAvg to accommodate such variability. Advances were also made in scheduling and client selection—since training with all clients each round is infeasible, efficient and fair sampling strategies became essential.

On the theoretical side, convergence analysis of federated SGD under non-IID conditions helped deepen understanding of FL dynamics. These innovations collectively improved the robustness, scalability, and efficiency of FL systems.

Adoption of FL spread beyond Google. In healthcare, Owkin built the first successful federated deep learning model for cancer diagnosis using data from distributed hospitals without sharing patient data [6]. This cross-silo FL approach enabled collaborative learning across institutions while preserving privacy.

Other domains followed suit: banks explored FL for fraud detection, and mobile carriers for joint network optimization. By 2021, FL was recognized as a general solution for scenarios where data sharing is constrained due to privacy, legal, or competitive concerns.

A landmark summary of FL's evolution was provided in the 2021 survey by Kairouz *et al.*, titled "Advances and Open Problems in Federated Learning" [1]. This 210-page survey cataloged key advances and identified open problems, including malicious client robustness,

fairness, efficiency, and expanding FL to new paradigms like decentralized training and federated reinforcement learning.

The emergence of dedicated FL workshops (e.g., IEEE FL Workshop, NeurIPS FL) further reflects the field's growth. Since the original FedAvg paper, FL has transitioned from a promising idea into a mainstream framework for privacy-aware collaborative AI.

1.5 Pre-FL Challenges and How FL Overcomes Them

Federated Learning (FL) was designed to overcome several limitations of prior approaches to distributed machine learning.

Privacy and Data Ownership: Traditional machine learning frameworks required centralizing data, which is infeasible for sensitive domains such as personal text messages or medical records. Before FL, organizations had to either avoid using such data or implement complex privacy-preserving techniques on centralized datasets. The core innovation of FL is that *raw data never leaves the client device*; only learned parameters or gradients are shared [1]. This fundamentally reduces privacy risks, aligns with user expectations, and supports compliance with data regulations through a "privacy by design" model—training occurs locally on user devices, and only aggregated updates are transmitted. This approach enables a form of anonymity and avoids the need for trusted central repositories, reducing the risk of data breaches.

Communication Efficiency: Conventional distributed ML assumed data already resided within a data center, avoiding bandwidth constraints. However, in FL, data originates on devices connected via unreliable or limited-bandwidth networks. Communication thus becomes a central bottleneck [2]. The FedAvg algorithm mitigates this by allowing each client to perform multiple local SGD steps before communicating with the server, drastically reducing the number of required communication rounds. Ongoing research builds on this with techniques like gradient compression, sparse updates, and adaptive communication scheduling. These innovations allow FL to trade local computation for reduced bandwidth usage—an ideal trade-off in edge environments.

Scalability to Many Clients: Classical distributed systems were built for tens or hundreds of data-center nodes, but FL must handle thousands to millions of clients. It is infeasible for all clients to participate in every training round. Instead, FL uses a server to sample a subset of clients each round and relies on asynchronous or semi-synchronous updates to tolerate stragglers. The scalability of FedAvg stems from this architecture—each round involves only a subset, yet all devices eventually contribute [1]. Additional scalability is achieved through hierarchical aggregation and load-balancing strategies adapted from distributed systems research.

Robustness to Heterogeneity: Unlike uniform data-center setups, edge devices vary widely in computational capacity, battery availability, and network connectivity. More critically, data across clients is often non-IID—for instance, one user's phone might have only pictures of dogs, while another's has only cats. Classical SGD assumes balanced, IID data, but FL algorithms like FedAvg are explicitly designed to operate under such non-IID conditions [2]. Surprisingly, this setting introduces a regularization effect when local updates are averaged. Further research has proposed client-specific personalization, update normalization, and tailored workloads for slower devices, enhancing FL's robustness to both system

and data heterogeneity.

In essence, FL shifts the paradigm from "bring data to the model" to "bring the model to the data." It integrates principles from stochastic optimization with modern systems design to support distributed, privacy-preserving learning on edge devices. By addressing the challenges of privacy, communication cost, scalability, and heterogeneity, FL enables learning on previously inaccessible data silos while respecting user constraints.

As Kairouz *et al.* emphasize, FL "mitigates many of the systemic privacy risks and costs resulting from traditional, centralized machine learning" [1]. Its rise is driven both by technological constraints—limited device bandwidth, battery, and compute—and growing societal demands for privacy-aware AI. As such, FL has become a cornerstone of modern distributed learning and privacy-preserving AI systems.

2 Problem of Non-IID Datasets

The central challenge addressed in this work is the presence of *non-independent and non-identically distributed (non-IID)* data across clients in Federated Learning (FL), which leads to degraded model performance, slower convergence, and increased optimization complexity.

2.1 Nature and Types of Non-IID Distributions

In Federated Learning, each client typically collects its own data based on individual behaviors, geographic locations, devices, or time windows, resulting in statistically diverse data distributions. Formally, accessing a data point involves first sampling a client $i \sim Q$, then sampling a data point $(x, y) \sim P_i(x, y)$ from that client's local distribution. The key source of heterogeneity is that $P_i \neq P_j$ for different clients.

This non-identicalness can take various forms:

- **Feature Distribution Skew (Covariate Shift):** The marginal distributions $P_i(x)$ differ across clients, even if $P(y|x)$ remains consistent. E.g., in handwriting recognition, users may write the same letters but with different styles.
- **Label Distribution Skew (Prior Probability Shift):** The marginal label distributions $P_i(y)$ vary even if the class-conditional $P(x|y)$ is the same. For example, users in different regions may encounter different objects or use different emojis.
- **Concept Drift (Same Label, Different Features):** The conditional $P_i(x|y)$ varies, meaning the same label may have different features on different clients (e.g., houses or fashion items vary across cultures).
- **Concept Shift (Same Features, Different Label):** The conditional $P_i(y|x)$ varies, implying the same features might be labeled differently due to personal preference or regional semantics (e.g., sentiment analysis).
- **Quantity Skew:** Clients may hold vastly different amounts of data, making client contributions highly imbalanced.

These forms of heterogeneity lead to biased local updates, inconsistent global models, and unpredictable convergence behavior.

2.2 Why Non-IID Data Is a Hard Problem

Non-IID data disrupts standard assumptions in SGD-based training:

- Local updates diverge from the global optimum, especially when each client's objective differs.
- Aggregation of non-aligned gradients causes instability and poor generalization.
- Classic convergence guarantees fail when the client datasets vary drastically in size and distribution.

This makes training a single global model across all clients difficult—sometimes even ill-posed—unless corrective strategies

3 Solving the Problem: Federated Averaging as a Baseline Approach

To address the problem of training models across massively distributed, privacy-sensitive, and non-IID client data, McMahan *et al.* (2017) [2] introduced **Federated Averaging (FedAvg)**. This algorithm allows clients to perform several local SGD updates before averaging model weights on a central server, significantly reducing communication costs. In this project, we aim to replicate their baseline experiments and evaluate FedAvg's performance under various non-IID conditions to establish a foundation for comparing more advanced techniques.

3.1 Federated Averaging Algorithm

The **Federated Averaging (FedAvg)** algorithm [2] is a fundamental technique in Federated Learning (FL), designed to address decentralized optimization with minimal communication overhead. In this setting, we assume there are K total clients (e.g., user devices), each holding a private local dataset $\mathcal{P}_k$ of size n_k. The global objective is to minimize a loss function over all client data:

$$\min_{w \in \mathbb{R}^d} f(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w), \quad \text{where } F_k(w) = \frac{1}{n_k} \sum_{i=1}^{n_k} \ell(w; x_i^k) \tag{1}$$

Here,

- w denotes the global model parameters,
- $F_k(w)$ is the local objective function for client k,
- $\ell(w; x_i^k)$ is the loss incurred on example x_i^k at client k,
- n_k is the number of data points at client k, and $n = \sum_{k=1}^{K} n_k$ is the total number of data points across all clients.

FedAvg optimizes this objective iteratively by selecting a random subset of clients in each round and allowing them to perform local updates on their private data. These updates are aggregated by a central server using a weighted average to form the new global model.

The main variables used in FedAvg include:

- K – total number of clients,
- C – fraction of clients sampled in each round ($0 < C \leq 1$),
- E – number of local training epochs performed on each client,
- B – local minibatch size,
- η – learning rate for local SGD.

The central server coordinates the training process, and each client performs local SGD over E epochs on its dataset. The server then aggregates client models based on the relative data sizes.

The detailed FedAvg algorithm is shown in Algorithm 1.

Algorithm 1 FederatedAveraging

Server executes:

Initialize global model w_0
for each round $t = 1, 2, \dots$ **do**
    $m \leftarrow \max(C \cdot K, 1)$
    $S_t \leftarrow$ random set of m clients
    for each client $k \in S_t$ **in parallel do**
        $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
    end for
    $m_t \leftarrow \sum_{k \in S_t} n_k$
    $w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$ ▷ Weighted aggregation
end for

ClientUpdate(k, w) ▷ Run on client k

$\mathcal{B} \leftarrow$ split local data $\mathcal{P}_k$ into batches of size B
for each local epoch $i = 1$ to E **do**
    for each batch $b \in \mathcal{B}$ **do**
        $w \leftarrow w - \eta \nabla \ell(w; b)$
    end for
end for
return w to server

3.2 Summary and Comparison with Synchronous Stochastic Gradient Descent

Federated Learning (FL) is a paradigm shift in machine learning, designed to enable model training directly on decentralized user devices while preserving data privacy. As modern mobile devices generate rich, sensitive, and high-volume data, transferring such data to central servers for training becomes infeasible due to privacy concerns and communication constraints. To overcome this, McMahan *et al.* [2] proposed **Federated Averaging (FedAvg)**, a novel optimization algorithm that enables collaborative learning without centralizing data.

The goal of FL can be expressed as minimizing a global loss function over a distributed dataset partitioned across K clients:

$$\min_{w \in \mathbb{R}^d} f(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w) \tag{2}$$

For client-partitioned data, this objective becomes:

$$f(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w), \quad \text{where } F_k(w) = \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w) \tag{3}$$

Here, $F_k(w)$ is the local objective on client k, n_k is the number of local samples, and $\mathcal{P}_k$ is the index set of those samples.

FedAvg improves model performance through iterative model averaging. Each round, a random subset of clients (fraction C) receives the current model, performs local training using E epochs of SGD with batch size B, and sends the updated model back to the server. The server aggregates these updates proportionally to n_k to produce a new global model.

Unlike traditional **synchronous SGD**, which requires communication after every mini-batch update (making it prohibitively expensive in federated settings), FedAvg achieves **10–100x reduction in communication rounds** by offloading more computation to the clients. This trade-off is well-justified because communication is the bottleneck in FL, not computation. Devices often have idle CPU/GPU capacity but are bandwidth-limited.

In summary, the key properties of Federated Learning include:

- **Non-IID Data:** Each client's data may follow a unique distribution, reflecting user-specific behavior.
- **Unbalanced Data:** Clients can have vastly different amounts of data.
- **Massive Distribution:** FL must scale to millions of clients with minimal coordination.
- **Communication Constraints:** Devices may have poor or intermittent connectivity.

These challenges make direct use of traditional distributed learning algorithms (such as synchronous SGD) inefficient and impractical in FL scenarios. FedAvg addresses these issues by:

- Allowing multiple local updates before communication.
- Aggregating model weights rather than raw gradients.
- Maintaining privacy by keeping data on-device.

Furthermore, the study shows that FedAvg is robust even under highly skewed and non-IID distributions, and performs well on deep models trained across image and language tasks. This makes it an ideal baseline for Federated Learning research and practice.

3.3 Experiments on Different Parameters (*click here for implementation*)

3.3.1 Model Architectures and Dataset Partitioning

Our initial study includes two model families applied to the MNIST digit recognition task [**?**]:

- **MNIST 2NN:** A simple multilayer perceptron (MLP) with two hidden layers, each containing 200 ReLU-activated units. This model contains a total of 199,210 parameters.

- **MNIST CNN:** A convolutional neural network with two convolutional layers using 5×5 kernels. The first layer has 32 channels and the second has 64 channels, each followed by 2×2 max pooling. This is followed by a fully connected layer with 512 ReLU-activated units and a final softmax output layer. The total parameter count is 1,663,370.

To study federated optimization, we explore two data partitioning strategies on the MNIST dataset across 100 simulated clients:

- **IID Partition:** The training data is randomly shuffled and divided evenly among 100 clients, with each client receiving 600 examples.

- **Non-IID Partition:** The data is sorted by digit labels and divided into 200 shards, each containing 300 examples. Each of the 100 clients is then assigned two shards, meaning most clients receive data from only two digit classes. This forms a pathological non-IID distribution designed to test the robustness of federated algorithms under extreme heterogeneity.

3.3.2 Effect of Client Fraction on Convergence Speed

In this section, we analyze the impact of increasing the degree of parallelism in federated training by varying the **client fraction** C. This parameter determines the proportion of total clients selected to participate in each communication round. For example, with $C = 0.1$, 10% of the total 100 clients (i.e., 10 clients) are sampled randomly per round.

We also define:

- E: The number of **local epochs**, i.e., how many times each client iterates over its local data per round.
- B: The **local mini-batch size** used during each client's SGD update.

We report the number of communication rounds required to reach a target test accuracy of 97% for the 2NN and 99% for the CNN model under both IID and non-IID settings. Table 1 below summarizes the results. Parentheses indicate the speedup factor relative to the baseline with $C = 0$.

As observed, increasing C has a marginal effect when using large batch sizes ($B = \infty$), but has a substantial impact when using smaller batch sizes (e.g., $B = 10$). Particularly in non-IID settings, using more clients per round accelerates convergence significantly. Based on these findings, we fix $C = 0.1$ for the remaining experiments, as it balances communication efficiency and convergence rate effectively.

2NN	IID		Non-IID	
C	$B = \infty$	$B = 10$	$B = \infty$	$B = 10$
0.0	1455	316	4278	3275
0.1	1474 (1.0×)	87 (3.6×)	1796 (2.4×)	664 (4.9×)
0.2	1658 (0.9×)	77 (4.1×)	1528 (2.8×)	619 (5.3×)
0.5	— (—)	75 (4.2×)	— (—)	443 (7.4×)
1.0	— (—)	70 (4.5×)	— (—)	380 (8.6×)
CNN, $E = 5$				
0.0	387	50	1181	956
0.1	339 (1.1×)	18 (2.8×)	1100 (1.1×)	206 (4.6×)
0.2	337 (1.1×)	18 (2.8×)	978 (1.2×)	200 (4.8×)
0.5	164 (2.4×)	18 (2.8×)	1067 (1.1×)	261 (3.7×)
1.0	246 (1.6×)	16 (3.1×)	— (—)	97 (9.9×)

Figure 1: Effect of client fraction C on communication rounds for MNIST 2NN and CNN. Speedups relative to $C = 0$ baseline are shown in parentheses.[2]

3.3.3 Effect of Increasing Local Computation

In this section, we fix the client fraction to $C = 0.1$ and analyze the effect of increasing local computation by varying the number of local epochs E and the mini-batch size B. This effectively controls the number of local updates performed on each client before aggregation.

We define the expected number of updates per client per round as:

$$u = \frac{nE}{KB}$$

where n is the total dataset size and K is the number of clients. By increasing u through larger E or smaller B, we aim to reduce the number of communication rounds while maintaining target accuracy.

Figure 2 clearly demonstrates that increasing local computation per client (i.e., larger E, smaller B, or both) significantly reduces communication rounds required to reach target accuracy. For IID MNIST, FedAvg with high local updates achieves up to 35× speedup over FedSGD. Even under the extreme non-IID partition, speedups range from 2.8× to 3.7×, confirming the robustness of the approach.

This finding indicates that increasing local computation is an effective way to mitigate communication overhead in federated settings without sacrificing model performance.

4 Simple Implementations of FedAvg, FedProx, FedAdam and FedYogi

This section presents pseudocode and brief explanations for three widely used federated optimization algorithms: **FedAvg**, **FedProx**, **FedAdam**, and **FedYogi**. These algorithms

MNIST CNN, 99% ACCURACY					
CNN	E	B	u	IID	NON-IID
FEDSGD	1	∞	1	626	483
FEDAVG	5	∞	5	179 (3.5×)	1000 (0.5×)
FEDAVG	1	50	12	65 (9.6×)	600 (0.8×)
FEDAVG	20	∞	20	234 (2.7×)	672 (0.7×)
FEDAVG	1	10	60	34 (18.4×)	350 (1.4×)
FEDAVG	5	50	60	29 (21.6×)	334 (1.4×)
FEDAVG	20	50	240	32 (19.6×)	426 (1.1×)
FEDAVG	5	10	300	20 (31.3×)	229 (2.1×)
FEDAVG	20	10	1200	18 (34.8×)	173 (2.8×)

Figure 2: Number of communication rounds to reach target accuracy for FedAvg vs. FedSGD on MNIST CNN (99%) and Shakespeare LSTM (54%). The expected updates per round $u = \frac{nE}{KB}$ is shown per configuration. Speedups relative to FedSGD baseline are shown in parentheses.[2]

vary in how they handle data heterogeneity, communication efficiency, and convergence on non-IID datasets.

4.1 FedAvg: Federated Averaging

FedAvg [2] is the foundational FL algorithm introduced in 2017 by McMahan *et al.*. It performs multiple local SGD steps on each client followed by averaging of model weights across selected clients. Although it is simple and communication-efficient, FedAvg suffers from slow convergence and instability when client data distributions are highly non-IID or unbalanced.

We use FedAvg as our baseline for comparison with more recent methods like FedProx and FedAdam.

4.2 FedProx: Federated Proximal

FedProx [11] is an enhancement of FedAvg designed to address data and system heterogeneity. In many real-world FL settings, clients differ not just in data distributions but also in system capabilities. FedAvg can become unstable when local client models diverge significantly from the global model due to non-IID data.

To combat this, FedProx introduces a **proximal term** in the client's local objective:

$$\min_w F_k(w) + \frac{\mu}{2}\|w - w_t\|^2$$

This regularization term keeps the local updates w close to the global model w_t, preventing client drift.

FedProx also supports variable local workloads and partial participation, making it robust under practical FL constraints. It has been shown to improve convergence in both convex and non-convex settings.

Pseudocode

Algorithm 2 FedProx

Initialize: global model w_0, proximal term coefficient μ
for each round $t = 1, 2, \ldots, T$ **do**
    Server selects clients $\mathcal{S}_t$
    for each client $k \in \mathcal{S}_t$ **in parallel do**
        Client receives w_t
        Minimize:

$$\min_w \left[F_k(w) + \frac{\mu}{2} \|w - w_t\|^2 \right]$$

        Client sends w_{t+1}^k to server
    end for
    Server aggregates:

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

end for

4.3 FedAdam: Adaptive Federated Optimization

FedAdam [12] is part of a family of adaptive optimization methods that apply well-established optimizers (like Adam, Yogi, or Adagrad) to the server update step in FL. The key idea is that while clients perform local training, the server behaves like a centralized optimizer and accumulates gradient statistics across rounds to improve convergence.

FedAdam maintains momentum (first moment) and variance (second moment) estimates on the server using client updates, then uses them to update the global model in an Adam-style adaptive fashion. It provides faster convergence than FedAvg, especially in heterogeneous environments with non-IID data.

FedOpt Framework: FedAdam is a specific case of the broader FedOpt framework proposed by Reddi et al., where the server optimization can be plugged in with different schemes (e.g., FedAdagrad, FedYogi).

Pseudocode

Algorithm 3 FedAdam

Initialize: w_0, $m_0 = 0$, $v_0 = 0$
for each round $t = 1, 2, \ldots, T$ **do**
  Server selects clients $\mathcal{S}_t$
  for each client $k \in \mathcal{S}_t$ **in parallel do**
    Client receives w_t and computes update $\Delta_k = w_{t+1}^k - w_t$
    Send Δ_k to server
  end for
  Aggregate:

$$\Delta_t = \sum_{k \in \mathcal{S}_t} \frac{n_k}{\sum_{j \in \mathcal{S}_t} n_j} \Delta_k$$

  Update moments:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\Delta_t, \quad v_t = \beta_2 v_{t-1} + (1 - \beta_2)\Delta_t^2$$

  Bias correction:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

  Update model:

$$w_{t+1} = w_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

end for

4.4 Algorithm Comparison

Algorithm	Heterogeneity Robustness	Server Effort	Client Complexity
FedAvg	No	Low	Low
FedProx	Yes	Low	Medium (proximal term)
FedAdam	Yes	High	Low

Table 1: Comparison of Federated Optimization Algorithms

4.5 Results

We present experimental results comparing the performance of FedAvg, FedProx, FedAdam and FedYogi on the SMS Spam/Not-Spam dataset and non-IID client setup.

4.5.1 FedAvg Results

Best scores on server-side tests:

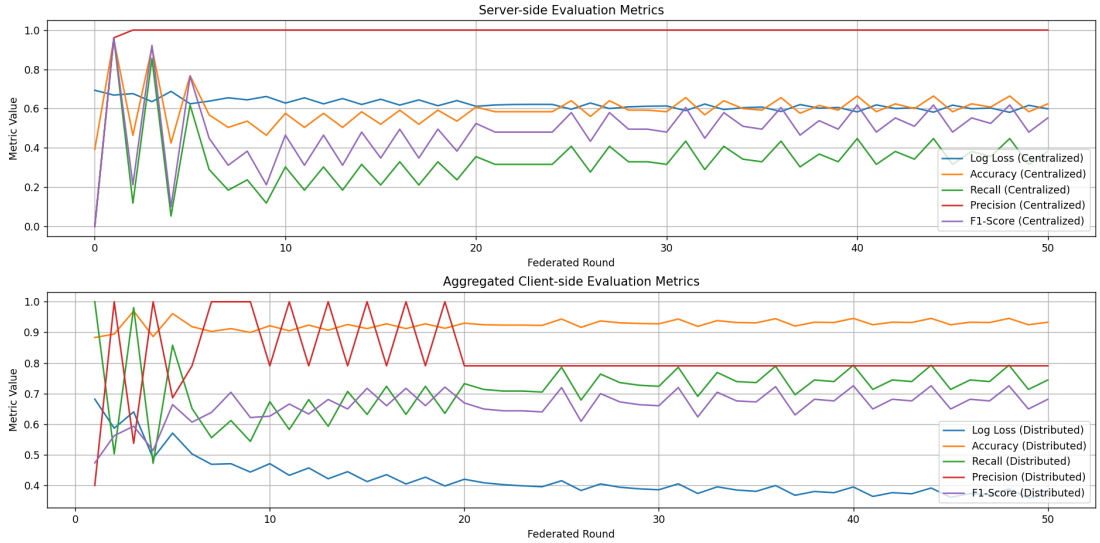


Figure 3: Fed-Avg Metrics

- Log-Loss: 0.5807891870680866
- Accuracy: 0.952
- Recall: 0.9605263157894737
- Precision: 1.0
- F1-score: 0.9605263157894737

4.5.2 FedProx Results

Best scores on server-side tests:

- Log-Loss: 0.6911164275736672
- Accuracy: 0.936
- Recall: 0.9473684210526315
- Precision: 1.0
- F1-score: 0.9473684210526315

4.5.3 FedAdam Results

Best scores on server-side tests:

- Log-Loss: 0.5109800063741935
- Accuracy: 0.896

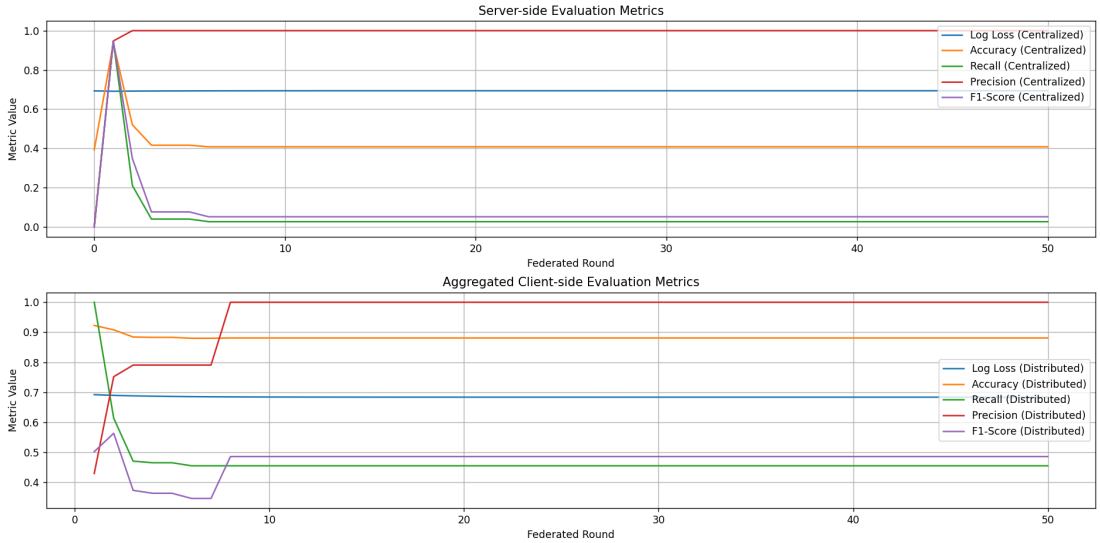


Figure 4: Fed-Prox Metrics

- Recall: 0.8421052631578947
- Precision: 1.0
- F1-score: 0.9078014184397163

4.5.4 FedYogi Results

Best scores on server-side tests:

- Log-Loss: 0.5414800279418301
- Accuracy: 0.944
- Recall: 0.9342105263157895
- Precision: 0.9841269841269841
- F1-score: 0.9530201342281879

5 Convergence Analysis of FedAvg on Non-IID Data

In this section, we provide a comprehensive convergence analysis of the Federated Averaging (FedAvg) algorithm in the presence of data heterogeneity (i.e., non-IID data distributions), as presented in the seminal paper by Li et al. [8]. We also highlight key theoretical insights, required assumptions, and practical implications.

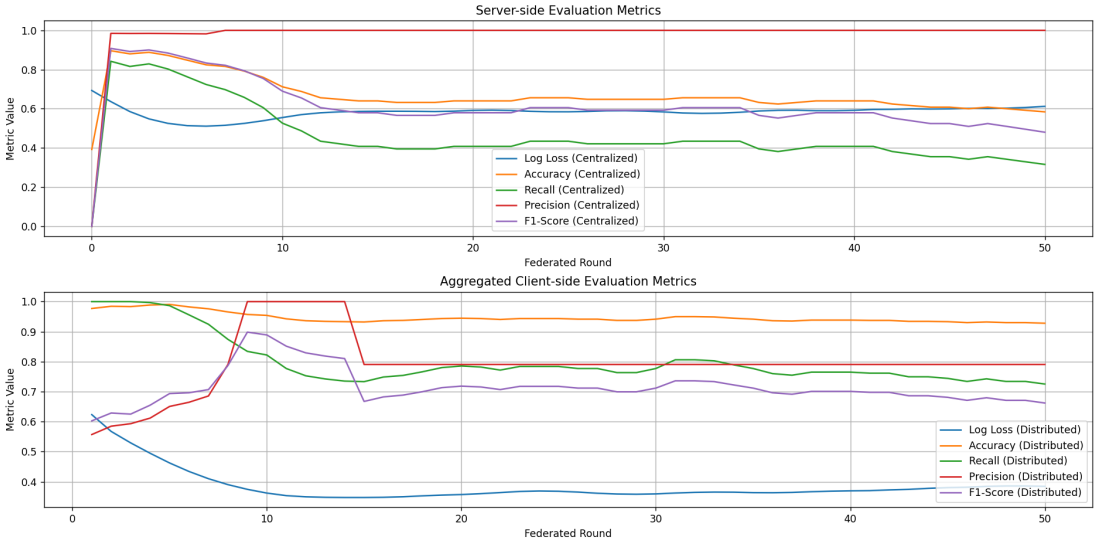


Figure 5: Fed-Adam Metrics

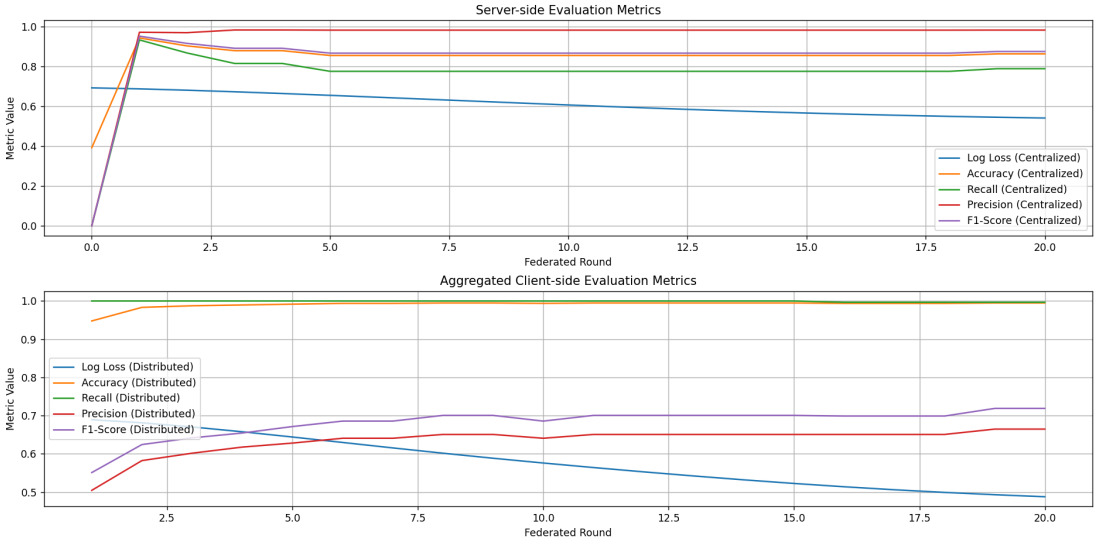


Figure 6: Fed-Yogi Metrics

Symbol	Description
w	Model parameter vector (global/shared model).
w_t	Global model after communication round t.
w_t^k	Local model of client k at the beginning of round t.
w_{t+1}^k	Updated local model on client k after E local SGD steps.
w_{t+1}	New global model after aggregation of local updates.
w^*	Optimal model that minimizes the global loss $F(w)$.
$\tilde{w}$	Stationary point reached by FedAvg with fixed step size.
$F(w)$	Global loss function: $F(w) = \sum_{k=1}^{N} p_k F_k(w)$.
$F_k(w)$	Local objective function at client k, defined as $F_k(w) = \frac{1}{n_k} \sum_{j=1}^{n_k} \ell(w; x_{k,j})$.
F^*	Optimal value of the global loss function.
F_k^*	Optimal value of the local loss function at client k.
$\ell(w; x_{k,j})$	Loss on the j-th data point of client k using model w.
p_k	Proportion of the dataset on client k ($p_k = \frac{n_k}{\sum_i n_i}$).
S_t	Subset of clients participating in round t.
E	Number of local SGD steps performed by each client.
η	Learning rate (step size) used for SGD updates.
η_t	Time-varying (decaying) learning rate used at round t.
κ	Condition number, defined as L/μ.
L	Smoothness constant: each F_k has L-Lipschitz gradients.
μ	Strong convexity constant: each F_k is μ-strongly convex.
σ_k^2	Variance of the stochastic gradient at client k.
G	Uniform upper bound on the norm of gradients across all clients.
Γ	Gradient dissimilarity across clients: $\Gamma = F^* - \sum_k p_k F_k^*$.
γ	Scaling term in the learning rate decay, $\gamma = \max\{8\kappa, E\}$.
T	Total number of global communication rounds.
B	Combined constant that upper bounds gradient noise, data heterogeneity, and local drift.

Table 2: Summary of Notation Used in FedAvg Convergence Analysis [8].

Notation Summary

5.1 Motivation and Setup

FedAvg [2] is a cornerstone algorithm in Federated Learning (FL) that alternates between local SGD updates at client devices and model averaging at a central server. However, most of its theoretical analyses historically assumed IID data and full client participation, which rarely hold in real-world FL setups.

Li et al. [8] break these assumptions and analyze FedAvg with:

- Non-IID data distributions.
- Partial client participation (only a subset of devices communicate per round).

- Strong convexity and smoothness of local objectives.

5.2 Problem Formulation

The global optimization problem is:

$$\min_w F(w) = \sum_{k=1}^{N} p_k F_k(w), \tag{4}$$

where p_k represents the relative size of the dataset on client k and $F_k(w) = \frac{1}{n_k} \sum_{j=1}^{n_k} \ell(w; x_{k,j})$ is the local empirical risk.

Each client performs E steps of SGD locally:

$$w_{t+1}^k = w_t^k - \eta \nabla F_k(w_t^k; \xi_t^k) \tag{5}$$

The server then aggregates the weights:

$$w_{t+1} = \sum_{k \in S_t} p_k w_{t+1}^k \tag{6}$$

5.3 Core Assumptions

- **Smoothness**: Each F_k is L-smooth.
- **Strong Convexity**: Each F_k is μ-strongly convex.
- **Bounded variance of gradients**: $\mathbb{E}[\|\nabla F_k(w, \xi) - \nabla F_k(w)\|^2] \leq \sigma_k^2$
- **Bounded dissimilarity**: Quantified using $\Gamma = F^* - \sum_k p_k F_k^*$ as in [8].

5.4 Main Result: Convergence Rate

The key result in the paper[8] shows FedAvg achieves a convergence rate of $\mathcal{O}(\frac{1}{T})$ under strongly convex objectives. The full convergence bound is:

$$\mathbb{E}[F(w_T)] - F^* \leq \frac{2\kappa}{\gamma + T} \left(\frac{B}{\mu} + 2L\|w_0 - w^*\|^2 \right), \tag{7}$$

where:

- $\kappa = L/\mu$ is the condition number.
- $B = \sum_k p_k^2 \sigma_k^2 + 6L\Gamma + 8(E-1)^2 G^2$.
- $\gamma = \max\{8\kappa, E\}$.
- G is a uniform bound on gradient norm.

Interpretation:

- Increasing E reduces communication but inflates the error due to local drift [8].
- A large Γ (strong non-IIDness) increases the error floor [8].
- Diminishing learning rate $\eta_t = \frac{2}{\mu(\gamma+t)}$ is essential for convergence [8].

5.5 Illustrative Example: Why Large E Can Hurt

Consider two clients with:

$$F_1(w) = (w-1)^2,$$
$$F_2(w) = 2(w-5)^2$$

Then the global objective is:

$$F(w) = 0.5F_1(w) + 0.5F_2(w) = (w-1)^2 + (w-5)^2 \tag{8}$$

- Global minimum: $w^* = \frac{11}{3}$
- With large E, client 1 converges to 1, client 2 to 5 $\Rightarrow$ global average $= 3 \neq \frac{11}{3}$
- Shows how local overfitting breaks global convergence [8].

Dataset	Description
`mnist balanced`	MNIST dataset partitioned equally among clients with uniform label distribution; simulates ideal IID setting.
`mnist unbalanced`	MNIST dataset where clients hold unequal numbers of samples and non-overlapping label subsets; simulates quantity and label skew.
`synthetic(0,0)`	Synthetic data generated with identical model parameters and identical distributions across clients; represents ideal IID baseline for algorithm validation.
`synthetic(1,1)`	Synthetic data with varying local model parameters and label distributions per client; used to simulate high heterogeneity (non-IID setting).

Table 3: Summary of datasets used in convergence analysis of FedAvg under different levels of non-IIDness.

5.6 Impact of Local Steps E

We empirically examine the role of local updates (E) in the convergence behavior of FedAvg. As shown in Figure 7, increasing E reduces the number of communication rounds required to reach a given accuracy up to a point. However, overly large E leads to model drift, especially on unbalanced datasets, degrading convergence.

- A small E leads to heavy communication but stable convergence.
- A large E reduces communication cost but introduces error due to local overfitting.
- On `mnist unbalanced` and `synthetic(1,1)` datasets, E shows a non-monotonic effect.
- This behavior aligns with theoretical bounds containing $(E-1)^2$ terms [8].

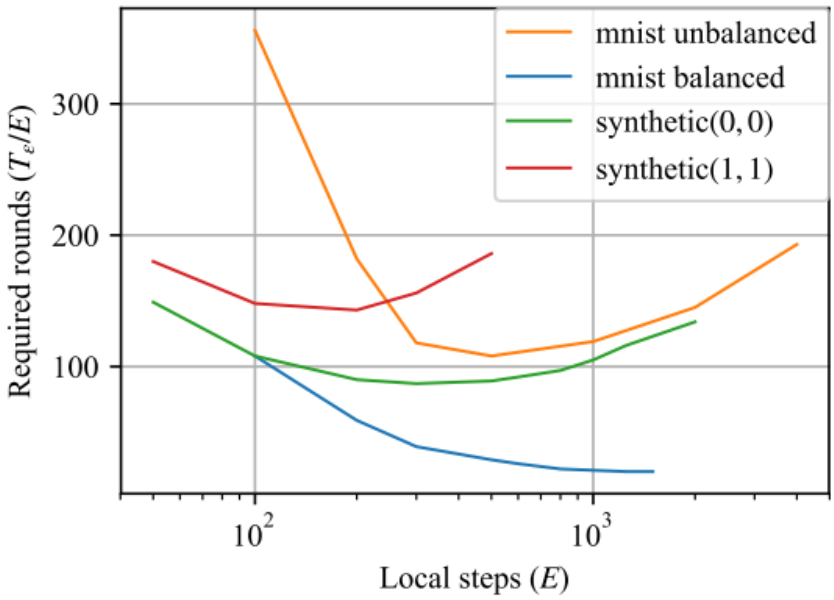


Figure 7: Effect of E on required communication rounds across datasets.

Scheme	Sampling	Averaging Formula	Convergence Ra
Original (McMahan et al., 2017)	$S_t \sim \mathcal{U}(N, K)$	$\sum_{k \notin S_t} p_k w_t + \sum_{k \in S_t} p_k w_t^k$	Not guaranteed
Scheme I (Sahu et al., 2018)	$S_t \sim \mathcal{W}(N, K, \mathbf{p})$	$\frac{1}{K} \sum_{k \in S_t} w_t^k$	$\mathcal{O}(1/T)$
Scheme II (Li et al., 2020)	$S_t \sim \mathcal{U}(N, K)$	$\sum_{k \in S_t} p_k \cdot \frac{N}{K} w_t^k$	$\mathcal{O}(1/T)$
T-Scheme II (Transformed)	$S_t \sim \mathcal{U}(N, K)$	Rescaled objectives: $F_k(w) \to \tilde{F}_k(w) = N p_k F_k(w)$	Same as Scheme I

Table 4: Sampling and Averaging Strategies in FedAvg with theoretical convergence guarantees.[8]

5.6.1 Impact of Number of Clients K

Figure 8 demonstrates how the number of participating clients per round (K) impacts convergence. While theory suggests that increasing K reduces variance (via a term $\mathcal{O}\left(\frac{EG^2}{K}\right)$), experiments reveal only a slight improvement in convergence.

- Larger K marginally improves convergence speed, especially in synthetic datasets.
- For $K \geq 50$, improvement plateaus.
- Minimal gains on `synthetic(0,0)` indicate diminishing returns of sampling more clients.
- Highlights trade-off between convergence and device availability.

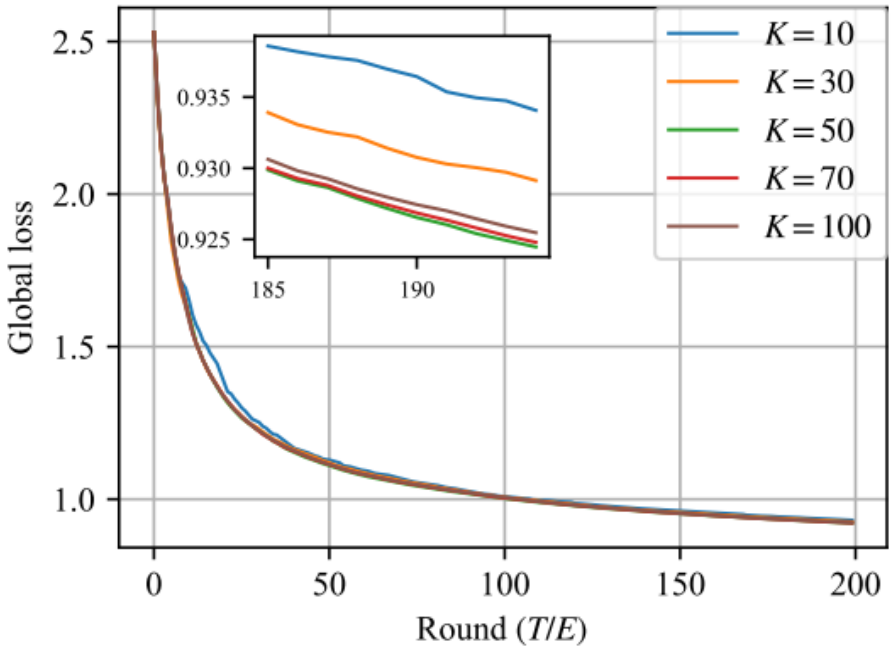


Figure 8: Effect of number of active clients K on FedAvg convergence.[8]

5.6.2 Effect of Sampling and Averaging Schemes

This experiment compares four sampling and averaging strategies. The results, visualized in Figures 9 and 9, show that sampling and aggregation have a direct impact on convergence rate and stability.

- In `mnist balanced`, both Scheme I and transformed Scheme II outperform the original FedAvg scheme.
- In `mnist unbalanced`, Scheme I provides the best convergence; Scheme II suffers instability.
- Transformed Scheme II is more stable but slower—indicating a trade-off.
- When data distributions p_k are uneven, careful scheme selection becomes critical.

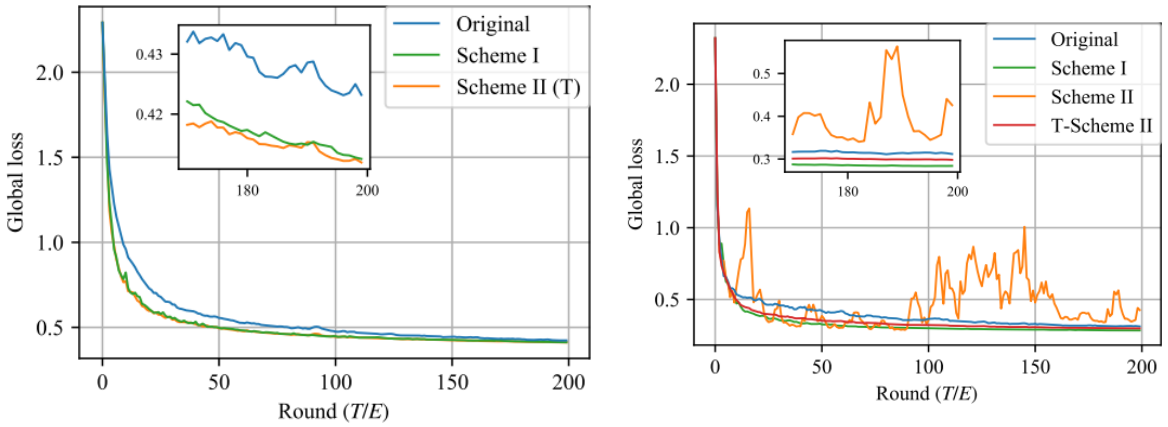


Figure 9: Comparison of averaging schemes on balanced and unbalanced datasets.[8]

5.6.3 Comparison of Averaging Schemes

To understand the effect of different client sampling and aggregation strategies on the convergence of FedAvg, we compare three popular schemes drawn from the literature:

- **McMahan et al. (2017)** uses uniform sampling $\mathcal{U}(N, K)$ with a hybrid update:

$$\sum_{k \notin S_t} p_k \mathbf{w}_t + \sum_{k \in S_t} p_k \mathbf{w}_t^k$$

- **Sahu et al. (2018)** employs weighted sampling $\mathcal{W}(N, K, \mathbf{p})$ and averages with:

$$\frac{1}{K} \sum_{k \in S_t} \mathbf{w}_t^k$$

- **Ours** adopts uniform sampling $\mathcal{U}(N, K)$ and reweights client contributions using:

$$\sum_{k \in S_t} p_k \frac{N}{K} \mathbf{w}_t^k$$

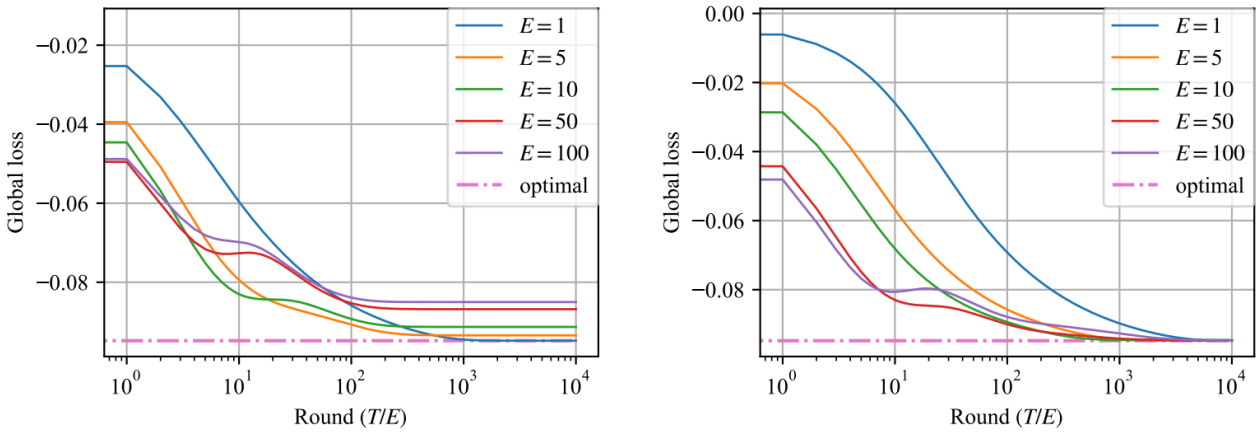
These schemes yield different convergence behaviors. While McMahan et al.'s method does not have formal convergence guarantees under non-IID data, both Sahu et al. and our approach achieve a theoretical rate of $\mathcal{O}(\frac{1}{T})$ under convex assumptions.

5.6.4 Theorem on Learning Rate Decay

Theorem 4: Fixed step size and $E > 1$ leads to non-convergence. Specifically,

$$\|\tilde{w} - w^*\|_2 = \Omega((E-1)\eta)\|w^*\|_2 \tag{9}$$

Implication: Without decaying η, FedAvg converges to a point at least $\Omega(\eta)$ away from optimum [8].



(a) Fixed learning rates

(b) Decayed learning rates

Figure 10: [8]

6 Implementation of Federated Learning: Gboard

A pioneering application of federated learning in the commercial domain was demonstrated by Google in their work on Gboard — the virtual keyboard used on over a billion Android devices. In the paper *"Federated Learning for Mobile Keyboard Prediction"* by Hard et al. (2019),[4] the authors train a recurrent neural network (RNN) language model directly on user devices using the FederatedAveraging (FedAvg) algorithm. The focus is on enabling next-word prediction in Gboard while maintaining user data privacy.

Prior to this work, Gboard relied on n-gram models trained using centralized, server-hosted logs data. This approach had limitations in privacy and representation, as server-side logs may not reflect the true typing behavior of users. The federated learning approach, in contrast, trains models using local data stored on client devices — without ever uploading raw data — thereby offering better alignment with user privacy and typing patterns.

To achieve this, the authors use a lightweight Coupled Input-Forgot Gate (CIFG) RNN architecture optimized for mobile environments, and implement on-device training through TensorFlow Lite. The results show that the federated model not only outperforms the baseline n-gram model in prediction recall, but also matches the performance of models trained on server data, demonstrating that privacy-preserving training can deliver commercial-grade quality. This work represents one of the earliest and most successful real-world implementations of federated learning in production systems.

6.1 Model Architecture and Technical Foundation

Prior research in mobile input systems has explored Finite State Transducers (FSTs) for decoding, correction, and prediction in keyboard interfaces. With the rise of neural methods, Recurrent Neural Networks (RNNs), especially Long Short-Term Memory (LSTM) models, have shown significant gains in language modeling accuracy and user experience. These mod-

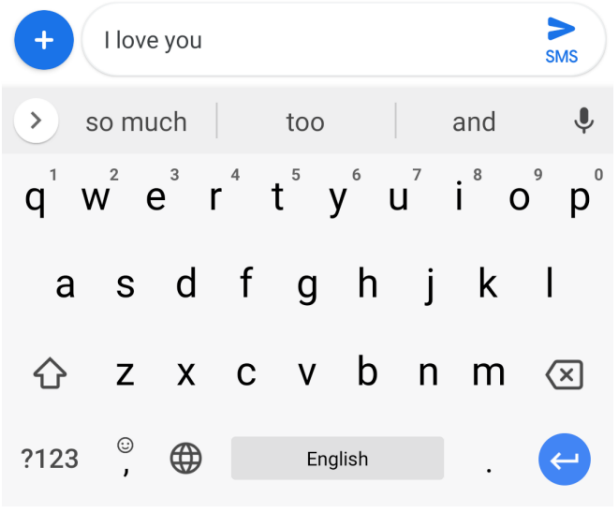


Figure 11: Next word predictions in Gboard. Based on the context "I love you", the keyboard predicts "and", "too", and "so much".[4]

els have been optimized for word prediction and keystroke savings under mobile constraints like low latency and limited memory.

Recent advancements in distributed training have been driven by the need for stronger privacy guarantees and data locality. Federated Learning (FL) extends server-based training to client devices, leveraging data stored locally. Prior works demonstrated FL with differential privacy, and Gboard has previously used FL to predict search queries based on typing context, although this work on next-word prediction marks the first full FL-based RNN language model deployed on a mobile keyboard [4].

The next-word prediction model used in Gboard is based on a compact and efficient RNN variant known as the Coupled Input-Forgot Gate (CIFG) [13]. CIFG simplifies the standard LSTM by tying the input and forget gates ($f_t = 1 - i_t$), thus reducing parameters by approximately 25% compared to traditional LSTMs. The model is implemented using TensorFlow, with inference handled via TensorFlow Lite.

To reduce memory footprint, the model uses tied input and output embedding matrices [14]. The network uses a vocabulary of $V = 10{,}000$ tokens, with an embedding and projection dimension $D = 96$, and a single-layer CIFG with 670 units. The total parameter count is 1.4 million, dominated by the embedding matrix, and the quantized on-device model is approximately 1.4MB in size.

6.2 Federated Learning Algorithm

Federated Learning (FL) is a decentralized training strategy that allows mobile devices (clients) to collaboratively train a shared model without transferring local data to a central server. Instead, clients compute updates on their own data and share model parameters with the server, which aggregates these to form an improved global model. This method offers enhanced privacy and supports data that is non-IID and unbalanced across devices.

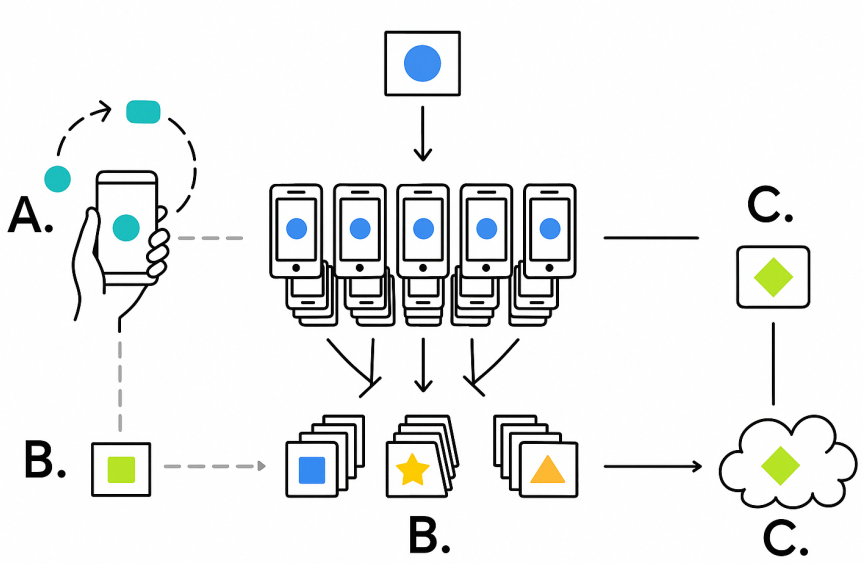


Figure 12: Illustration of the Federated Learning process. (A) Client devices compute local model updates using Stochastic Gradient Descent (SGD) on locally stored data. (B) A central server aggregates the client updates to form a new global model. (C) The updated global model is sent back to clients. This loop continues iteratively across training rounds [4].

FederatedAveraging (FedAvg) Procedure

The training proceeds in communication rounds:

1. At each round t, the server sends the current global model $\mathbf{w}_t$ to a randomly selected subset $\mathcal{K}$ of clients.

2. Each client $k \in \mathcal{K}$ updates the model locally using its dataset of n_k examples by performing SGD:
$$\mathbf{w}_{t+1}^k = \mathbf{w}_t - \epsilon \cdot \nabla F_k(\mathbf{w}_t) \tag{10}$$
where ϵ is the client learning rate, and ∇F_k is the average gradient on client k's data.

3. The server aggregates the client-updated models using a weighted average:
$$\mathbf{w}_{t+1} = \sum_{k \in \mathcal{K}} \frac{n_k}{N} \cdot \mathbf{w}_{t+1}^k \tag{11}$$
where $N = \sum_{k \in \mathcal{K}} n_k$ is the total number of examples across participating clients.

Privacy Considerations

This federated approach provides privacy benefits by ensuring raw user data never leaves the device. Updates are ephemeral and discarded after server-side aggregation, adhering to the principle of data minimization. While FL assumes a trustworthy aggregation server, it is often combined with secure aggregation protocols and differential privacy to further protect user information.

6.3 Training and Evaluation of the CIFG Language Model

This section describes the training of the CIFG (Coupled Input-Forgot Gate) next-word prediction model using both server-based and federated learning strategies. Performance is evaluated using logs data, client-held cache data, and live production experiments.

6.3.1 Server-based Training with Logs Data

Server-side training utilizes anonymized logs from users who have opted to share text snippets from Google apps. The dataset consists of approximately 7.5 billion sentences (avg. length: 4.1 words), primarily from chat applications.

- Training uses asynchronous SGD with learning rate $\eta = 10^{-3}$.
- No momentum or weight decay is applied.
- Batch size is 50 sentences.
- Network converges after 150 million steps.

Table 5: Logs data distribution by app type

App Type	Share
Chat	60%
Web Input	35%
Long-form Text	5%

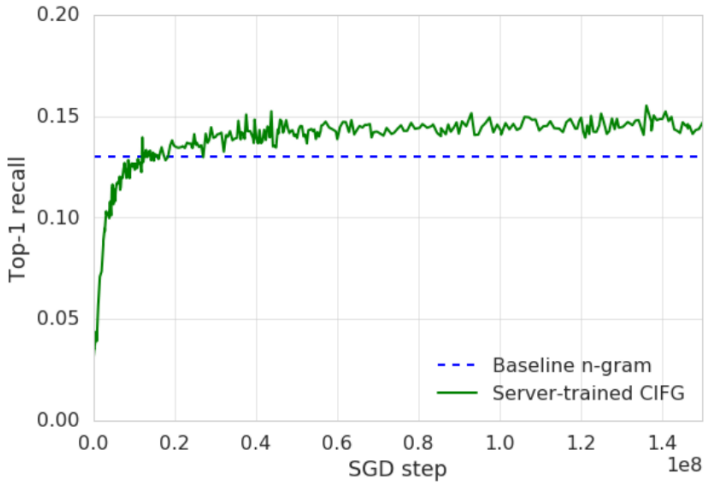


Figure 13: Top-1 recall of the CIFG model over SGD steps during server-based training, compared with the static n-gram baseline.[4]

6.3.2 Federated Training with Client Caches

The federated setup trains the model using on-device caches that contain local text and decoder prediction candidates.

Client Eligibility Criteria

- At least 2GB of available memory.
- Device is charging, idle, and on an unmetered network.
- Device is located in North America, running Gboard 7.3+ with US English enabled.

Federated Training Configuration

- Each training round aggregates updates from 100–500 clients.
- Momentum optimizer with Nesterov acceleration:
  - Momentum coefficient: 0.9
  - Server learning rate: 1.0
- Each client processes around 400 sentences per epoch.
- The CIFG model converges after 3000 rounds (4–5 days), covering 600M sentences across 1.5M clients.

Table 6: Client cache data distribution by app type

App Type	Share
Chat	66%
Social Media	16%
Web Input	5%
Other	12%

Evaluation Strategy

Unlike server-side training which uses explicit dataset splits, federated evaluation defines computation tasks for training, testing, and evaluation across a large population. The low probability of client reuse across tasks ensures independence of evaluation.

Comparison with N-gram Baseline

Although the n-gram baseline is not retrained, its recall is measured using decoder candidates stored in the training cache and compared against actual user-entered text.

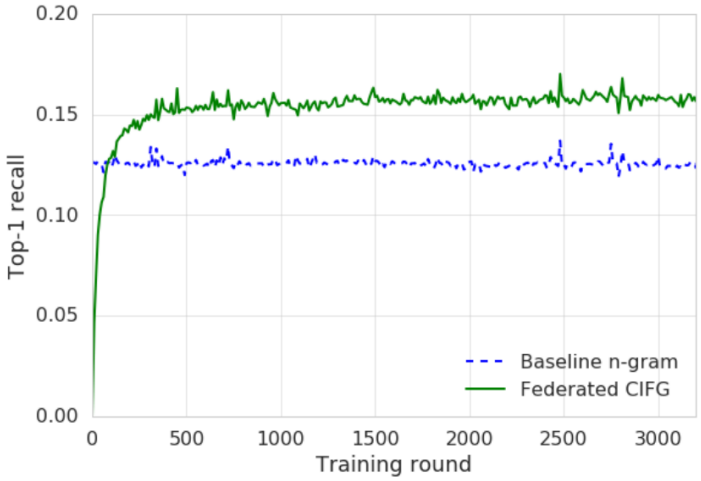


Figure 14: Top-1 recall of the CIFG during federated training, compared with the n-gram decoder performance on client cache evaluation.[4]

Top-1 Recall

Top-1 Recall is a performance metric used to evaluate next-word prediction models. It measures the percentage of times the model's highest-probability (top-1) prediction matches the actual next word typed by the user. Formally:

$$\text{Top-1 Recall} = \frac{\text{Number of correct top-1 predictions}}{\text{Total number of prediction instances}} \times 100\% \tag{12}$$

This metric is particularly important in keyboard applications like Gboard, where users typically select from the first suggested word. A higher Top-1 Recall indicates more accurate and useful predictions in real-time usage.

6.4 Evaluation Results: Federated vs. Server Training

The performance of the CIFG language model is compared across three evaluation sources:

- **Server-hosted logs data**
- **Client-owned caches**
- **Live user production experiments**

Recall Metrics

Model performance is evaluated using the Top-1 and Top-3 recall:

- **Top-1 Recall:** Percentage of times the model's top prediction matches the user's typed word.
- **Top-3 Recall:** Recall across the top three predicted candidates, reflecting the Gboard interface.

Table 7: Prediction recall on server-hosted logs data

Model	**Top-1 Recall**	**Top-3 Recall**
N-gram	13.0%	22.1%
Server CIFG	16.5%	27.1%
Federated CIFG	16.4%	27.0%

Table 8: Prediction recall on client-owned caches

Model	**Top-1 Recall (%)**
N-gram	12.5 ± 0.2
Server CIFG	15.0 ± 0.5
Federated CIFG	15.8 ± 0.3

Live Production Metrics

Table 9: Prediction impression recall (live experiments)

Model	**Top-1 Recall**	**Top-3 Recall**
N-gram	5.24% $\pm$ 0.02	11.05% $\pm$ 0.03
Server CIFG	5.76% $\pm$ 0.03	13.63% $\pm$ 0.04
Federated CIFG	**5.82%** $\pm$ 0.03	**13.75%** $\pm$ 0.03

Key Observations

- Both CIFG models outperform the n-gram baseline across all recall metrics.
- Federated CIFG achieves slightly better Top-1 recall on client caches (+0.8% absolute over server CIFG).
- In live user tests, federated CIFG shows a 1% relative improvement in both Top-1 and Top-3 impression recall.
- Click-through rate (CTR) for federated CIFG is 10% higher than the n-gram baseline.

References

[1] Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., ... & Zhao, S. (2021). *Advances and Open Problems in Federated Learning*. ar5iv. https://ar5iv.org/abs/1912.04977

[2] McMahan, H. B., Moore, E., Ramage, D., Hampson, S., & y Arcas, B. A. (2017). *Communication-efficient learning of deep networks from decentralized data*. arXiv. https://arxiv.org/abs/1602.05629

Table 10: Prediction click-through rate (CTR)

Model	**CTR (%)**
N-gram	2.13 ± 0.03
Server CIFG	2.36 ± 0.03
Federated CIFG	2.35 ± 0.03

[3] Konečný, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., & Bacon, D. (2015). *Federated Optimization:Distributed Optimization Beyond the Datacenter*. arXiv. https://arxiv.org/abs/1511.03575

[4] Hard, A., Rao, K., Mathews, R., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C., & Ramage, D. (2018). *Federated Learning for Mobile Keyboard Prediction*. Google Research. https://research.google/pubs/federated-learning-for-mobile-keyboard-prediction-2/

[5] Google AI Blog. (2019). *Federated Learning with Formal Differential Privacy Guarantees*. https://research.google/blog/federated-learning-with-formal-differential-privacy-guarantees/

[6] Owkin. (2020). *Story of the 1st Federated Learning Model at Owkin*. https://www.owkin.com/blogs-case-studies/story-of-the-1st-federated-learning-model-at-owkin/

[7] Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J., & Su, B.-Y. (2014). *Scaling Distributed Machine Learning with the Parameter Server*. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)* (pp. 583–598). USENIX Association. https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-li$_m$*u.pdf*

[8] Li, X., Huang, K., Yang, W., Wang, S., & Zhang, Z. (2020). *On the Convergence of FedAvg on Non-IID Data*. In *International Conference on Learning Representations (ICLR)*. https://openreview.net/pdf?id=HJxNAnVtDS

[9] Stich, S. U. (2019). *Local SGD Converges Fast and Communicates Little*. In *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/1805.09767

[10] Wang, J., & Joshi, G. (2019). *Cooperative SGD: A Unified Framework for the Design and Analysis of Communication-Efficient SGD Algorithms*. In *22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*. https://arxiv.org/abs/1804.05296

[11] Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., & Smith, V. (2020). *Federated Optimization in Heterogeneous Networks*. MLSys. arXiv:1812.06127

[12] Reddi, S. J., Charles, Z., Zaheer, M., Garrett, Z., Rush, K., Konečný, J., Kumar, S., & McMahan, H. B. (2021). *Adaptive Federated Optimization*. ICLR. arXiv:2003.00295

[13] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A Search Space Odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2017. Available: `https://ieeexplore.ieee.org/document/7508408`

[14] O. Press and L. Wolf, "Using the Output Embedding to Improve Language Models," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, Valencia, Spain, 2017, pp. 157–163. Available: `https://aclanthology.org/E17-2025/`