Shared Spotlight Meridian: Distributed Sparse Pseudorandom Functions for Scalable Federated Learning

Youlong Ding[†*], Peihua Mai[‡*], Jingqi Zhang[‡], Sherman S. M. Chow[§✉], Minxin Du[‖✉], Yan Pang[‡]
[†]*The Hebrew University of Jerusalem* [‡]*National University of Singapore*
[§]*The Chinese University of Hong Kong* [‖]*The Hong Kong Polytechnic University*
youlong.ding@mail.huji.ac.il, {peihua.m, zhangjingqi}@u.nus.edu,
smchow@ie.cuhk.edu.hk, minxin.du@polyu.edu.hk, jamespang@nus.edu.sg

Abstract—**Secure federated learning enables multiple clients to train a shared model while keeping raw data private. Minimizing communication is natural in secure multiparty computation, yet cryptographic mechanisms create tension. Consequently, existing secure aggregation protocols are ill-suited to high-dimensional sparse updates and forfeit sparsification gains, inflating communication by orders of magnitude relative to plaintext aggregation.**

Seeking efficiency under privacy, we introduce distributed sparse pseudorandom functions. Hidden alignment comes from a secretly shared spotlight index that illuminates the chosen coordinate and serves as a meridian that anchors aggregations of nonzero entries. Enabled by our cryptographic advances, we present a secure aggregation protocol with near-optimal client communication. Relative to a plaintext baseline, each client sends at most one extra bit per nonzero gradient element. Multi-server security holds unless all servers collude. At the client side, computational overhead is small, and server communication is optimal. Near-baseline accuracy is seen across computer vision, natural language processing, and recommendation experiments, with plaintext-level bandwidth savings.

1. Introduction

Secure collaboration maintains strict data boundaries. Having become pervasive, machine learning is increasingly paired with cryptographic solutions to ensure data privacy. Examples include distributed settings where data owners seek to train a shared model without revealing raw data. Rather than centralizing data, federated learning (FL) enables clients to train a model without sharing local data. Model aggregation in FedAvg [1] averages client updates to produce the global model, while training data stays local. Adoption has increasingly been driven by privacy regulations [2] and the need to learn from decentralized data. Nonetheless, communication overheads pose obstacles.

So far, large-scale deployments of FL remain limited. However, Google's Gboard deployment [3] demonstrates the feasibility of private FL, albeit via differential privacy. Existing cryptographic approaches face a client-bandwidth bottleneck, so reducing per-round communication is crucial. Risks to privacy are also intensifying since it has been shown that FL alone does not guarantee the privacy of training data. Model updates can be exploited by attacks [4], [5], [6]. Altogether, these pressures motivate two core questions. Necessarily, the two questions should be treated in tandem.

- *"How to reduce client communication in FL?"* Several methods improve communication efficiency, such as quantization [7], [8] and gradient sparsification [9], [10]. Many such optimizations leave data privacy unaddressed. Combining them with standard cryptographic methods often breaks compatibility. Hiding values and zero locations for privacy typically requires padding every coordinate, even when gradients are sparse. Once the server cannot learn which entries are zero, sparsity yields no communication savings. With these constraints, many optimizations are ineffective in private settings.
- *"How to ensure client privacy?"* Security research explored single-server secure aggregation, treating the machine learning (ML) process as a black box. Since a 4-round protocol for one aggregation was proposed [11], subsequent work [12], [13], [14], [15], [16], [17] has pursued stronger guarantees, typically without integrating ML-side communication optimizations.

Simply put, advances in FL and cryptography rarely interoperate, so gains in one seldom carry over to the other.

Security–ML Co-design: New Cryptographic Primitives for Efficient FL. Starting from this gap, we initiate a co-design of secure protocols and FL algorithms. Motivated by modern FL structures and optimization, we introduce cryptographic primitives tailored to these patterns. Consequently,

[*] Similar contribution level; [✉] Corresponding: Not all authors are responsible for proofreading. Please address error reports to corresponding authors.

[†] Ding was supported in part by a grant from the Israel Science Foundation (ISF Grant No. 1774/20), and by the European Union (ERC, SCALE,101162665). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.
[§] Chow was supported in part by the General Research Funds (CUHK 14210825, 14210621) from the Research Grant Council, the Hong Kong Special Administrative Region of the People's Republic of China; and by Direct Grant (4055238) and the Strategic Impact Enhancement Fund (3135517) from CUHK.
[‖] Du was supported in part by the HK PolyU Start-up Fund (P0052722).

we push the efficiency limits of private FL.

- On the machine-learning front, we identify gradient sparsification as a scalable mechanism to reduce client communication. It offers fine-grained, *per-client* control of communication, from full gradients to near zero, adaptive to each client's instantaneous network conditions and *independent of the global model size*.
- On the cryptography front, the pseudorandom function (PRF) is a versatile cornerstone. We introduce *distributed sparse pseudorandom* functions (DSPRFs), a novel PRF notion that reconciles sparsification with secure aggregation and achieves near-optimal client communication.

Technical Overview. DSPRFs resolve the sparsification–privacy mismatch. They enable clients to transmit sparse gradient updates while maintaining cryptographic privacy. The key innovation is that each client can secretly *share* knowledge of a single gradient coordinate (the *spotlight*) among servers by sending each a DSPRF distributed key. This shared coordinate serves as an implicit alignment reference (the *meridian*) so the servers can recover the sum of sparse gradients without learning the coordinate.

The only information about the coordinate that reaches the servers is carried in these per-server distributed key shares. DSPRFs guarantee that each distributed key share is indistinguishable from random; no proper subset of servers can infer the coordinate, and only if *all* collude is it revealed. Specifically, DSPRFs feature two evaluation modes, each tied to a distinct key type.

- **Core evaluation mode**: This client-side mode encrypts a sparse gradient g under a master key. It outputs a pseudorandom mask r and a (pseudorandom) coordinate i. The client encrypts g at coordinate i[1], producing a *scalar* ciphertext $g + r$ broadcast to all servers.
- **Distributed evaluation mode**: This server-side mode uses each distributed key share to run the evaluation. Each evaluation outputs two vectors that are additive shares of a vector encoding of i and of the mask r. With only local computation on the scalar ciphertext $g+r$, each server maps it to the hidden coordinate using its index share and subtracts its mask share to obtain its additive share of the client update. Summing these shares across servers recovers the aggregated sparse gradient, while no proper subset learns g or i.

Feature Highlights. Powered by DSPRFs, our protocol lets a client, after generating and distributing keys to multiple servers, participate in any number of FL rounds. It features:

- *Near-optimal client communication cost.* The client can use any gradient sparsification method with any sparsification ratio. For each selected gradient element $(i, g_i) \in [N] \times \mathbb{Z}_q$, the client sends at most $\log N + \log q + 1$ bits, only one bit more than plaintext.
- *Small client computation overhead.* Encrypting each gradient element uses only one AES[2] call, an inner product as in cryptosystems based on the learning-with-error (LWE) assumption, and an incremental invocation of a local pseudorandom generator (PRG) (Definition 5).
- *Optimal server communication cost.* Each server performs only local computation to obtain its additive share of the summed gradients. Server computation overhead remains practical. Beyond AES and an LWE inner product, the server performs homomorphic secret sharing (HSS) evaluation (Definition 7) for a low-degree (degree-5) computation, which is efficient in practice.

TABLE 1. COMPARING FL PROTOCOLS (n: NUMBER OF NONZERO GRADIENT ENTRIES, $\theta \in \mathbb{Z}_q^N$: MODEL PARAMETERS)

	Client Communication (per round)	# Servers
Plaintext	$n \cdot (\log N + \log q)$	1
Flamingo [18]	$\Omega(N \cdot \log q)$	1
e-SeaFL [13]	$\Omega(N \cdot \log q)$	2
Ours	$n \cdot (\log N + \log q + 1)$	≥ 2

Table 1 compares our scheme with the state of the art.

Related Work. To protect client privacy in FL, the seminal work [11] and subsequent work [19], [20] use pairwise masks to hide individual updates from the server, but require multiple rounds of interaction per iteration. An alternative is multi-key fully-homomorphic encryption (MK-FHE), which lets clients encrypt under their own keys while the server sums ciphertexts [21]; this offers strong privacy at the cost of substantial computation and communication.

Flamingo [18] is a single-server FL system that removes costly per-round setup by allowing reuses of cryptographic secrets across rounds. To handle client dropouts, Flamingo uses designated decryptor clients to assist in mask removal, reducing client-server interactions. However, like most secure multiparty computation approaches [13], it is not optimized for sparse updates, so its communication resembles that of dense vectors and cannot exploit plaintext sparsity.

A complementary line distributes trust across two non-colluding servers. Prio [22] and Poplar [23] employ (2-party) distributed point functions (DPFs) [24] to aggregate private statistics. DPFs natively yield a secret sharing of a unit vector. Direct extensions to sparse vectors incur nontrivial overhead, which limits practicality in communication-sensitive settings. Table 2 compares our approach with alternatives that target low communication for sparse gradients.

We remark that non-colluding servers have also supported more complex ML tasks, such as decision-tree inference [27] and training [28].

2. Problem Statement and ML Background

Target Scenario with Parameters. The parameters are as follows. $\lambda = 128$ is the security parameter. $\boldsymbol{\theta} \in \mathbb{Z}_q^N$ is the machine-learning model. N is the number of parameters, typically ranging from 2^{10} to 2^{40}. (The largest current models, such as GPT, are estimated to have 2^{40} parameters.) $q = 2^{32}$ is the model precision. Let n_c be the number of

1. Here, we assume i matches the coordinate that the client will use.
2. Advanced encryption standard, or any pseudorandom permutation

TABLE 2. Client communication cost and computation overhead (per gradient element) for each round of FL with sparse gradients (λ is the security parameter, n_c is the total number of clients, $\theta \in \mathbb{Z}_q^N$ is the ML model.)

	Client Communication	Client Computation Overhead	# Servers
Plaintext (static sparsity)	$\log q$	N/A	1
Plaintext (dynamic sparsity)	$\log N + \log q$	N/A	1
MK-FHE	$\Omega_\lambda(n_c \cdot (\log N + \log q))$	$\Omega(N)$ FHE operations	1
2-Party DPF [25]	$\lambda \cdot \log N + \log q$	$\log N$ AES	2
Multiparty DPF [26]	$\Omega(\lambda \cdot \sqrt{N})$	$\Omega(\sqrt{N})$ AES	$\geq$3
Ours (static sparsity)	$\underline{\log q}$	1 AES + 1 LWE	$\geq$2
Ours (dynamic sparsity)	$\underline{\log N + \log q + 1}$	1 local PRG + 1 AES + 1 LWE	$\geq$2

clients, with $n_c \geq 10$, n_s be the number of servers, with $n_s \geq 2$, and let T be the number of federated learning iterations, with $T \geq 100$. They are all unbounded.

Communication Model. Each client has private, authenticated channels to servers and a broadcast channel to all servers. We assume reliable transport, where packet loss and retransmission are handled at the network layer. There is no client-to-client messaging. Clients are heterogeneous and may drop out due to device or network issues.

Dynamic Client Participation. We model the FL system as an unbounded sequence of training rounds. Any prospective participant may join at any moment. In every round, each registered client is free to decide independently whether to participate in the current round or skip the round.

Security Assumptions and Guarantees. We consider a *semi-honest* setting, in which all parties, *i.e.*, clients and servers, follow the protocol as specified, which is required for correctness. Robustness against malicious behavior, *e.g.*, via zero-knowledge proofs [29] and authentication, is left to future work. An adversary may corrupt an arbitrary set of clients and a proper subset of servers, possibly adaptively. Privacy holds as long as at least one server is not corrupted by the adversary. Individual client updates remain hidden, and the adversary learns only the prescribed aggregate and public metadata. These guarantees compose across rounds and tolerate arbitrary client dropouts.

Our security relies on standard assumptions: the hardness of LWE, the pseudorandomness of PRFs, and the security of the DPF and HSS used in our DSPRF constructions. This multi-server architecture is widely adopted in big data analytics, such as private statistics [22], [23], and in (outsourced) collaborative training [30], due to its scalability and efficiency compared to single-server counterparts.

Federated Learning. The goal of FL is to find a global model $\boldsymbol{\theta}$ using private data distributed across different clients by an iterative process. We recall federated averaging (FedAvg) [1] as follows.

1) Each (selected) client $c \in \mathcal{AC} \subseteq [n_c]$ locally optimizes its model $\boldsymbol{\theta}_c$ on its private data X_c with the objective $J(\cdot)$, and sends $\boldsymbol{\theta}_c$ to the server.
2) The server computes the global model $\boldsymbol{\theta}_{\text{avg}}$ as the average of client models $\{\boldsymbol{\theta}_c\}_{c \in \mathcal{AC}}$ with
$$\boldsymbol{\theta}_{\text{avg}} := \frac{1}{|\mathcal{AC}|} \sum_{c \in \mathcal{AC}} \boldsymbol{\theta}_c,$$
3) The server sends $\boldsymbol{\theta}_{\text{avg}}$ to all clients.
4) Repeat Steps 1–3 until convergence.

Gradient Sparsification (dynamic sparsity pattern). To reduce the high communication cost per training round, especially for large models, gradient sparsification has become a common approach [9], [10], [31], [32]. Instead of transmitting full-precision model weights or gradients, each client selects and transmits only a small subset of gradient coordinates, *e.g.*, top-k entries by magnitude.

Embedding Models (static sparsity pattern). Many modern machine-learning models, such as those in recommendation systems and large language models (LLMs), use embedding layers to map high-cardinality categorical inputs into continuous vector representations. Each discrete token or ID is assigned a unique embedding vector that is learned during training. Since only a small subset of tokens appears in each minibatch, the gradients for the embedding layer are sparse, with only the embeddings for observed tokens receiving nonzero updates. This sparsity pattern is predictable and intrinsic, and has been exploited in system optimizations and adversarial analysis. Embedding layers can dominate model parameter size, especially in domains with large vocabularies, making them critical for both computational and privacy considerations.

Secure Aggregation. Secure aggregation protocols [11] let the server learn only the sum of client updates, protecting individual contributions against a semi-honest server. Applying existing schemes to sparse updates is challenging. Naïve approaches often neglect sparsity and incur high overhead, motivating designs that preserve both efficiency and privacy.

In our setup, computing the exact aggregate could be overkill. Modern training is robust to small amounts of noise. Our goal is to secure the FL protocol in Figure 1 while tolerating a small, quantifiable approximation error.

3. Cryptography Preliminaries

Learning with Errors. The LWE assumption [33] with parameters (n, q, B) states that, given samples $(\mathbf{a}_i, \langle \mathbf{a}_i, \mathbf{s} \rangle +$

Plaintext FL Protocol

Public Parameters: Total number of model parameters $N \in \mathbb{Z}$, gradient group $\mathbb{Z}_q$, number of servers n_s, number of clients n_c, a training algorithm $\mathsf{Train}\colon \mathbb{Z}_q^N \times \mathcal{D} \to \mathbb{Z}_q^N$, a gradient sparsification algorithm $\mathsf{Sparse}\colon \mathbb{Z}_q^N \times \mathbb{Z} \to \mathbb{Z}_q^N$, total number of rounds T.

Inputs: Each client c holds local data X_c, static sparsity index set $\mathcal{SI}_c \subseteq [N]$, and a bandwidth bound B_c for $c \in [n_c]$. The server has an initial model $\boldsymbol{\theta}_1 \in \mathbb{Z}_q^N$.

- For round $t = 1, \dots, T$:
  - Client $c \in [n_c]$:
    - $*$ If not registered, choose whether to register in the FL system. If registering, send $\mathcal{SI}_c$ to the server.
    - $*$ If registered, choose whether to participate in the current round or simply skip.
  - Server:
    - $*$ Sample a batch of clients $\mathcal{AC}_t \subseteq [n_c]$ from the set of active clients.
  - Client $c \in \mathcal{AC}_t$:
    - $*$ $\vec{\boldsymbol{g}}_{t,c} \leftarrow \mathsf{Train}(\boldsymbol{\theta}_t, X_c) \in \mathbb{Z}_q^N$.
    - $*$ $\mathcal{I} \leftarrow \mathsf{Sparse}(\vec{\boldsymbol{g}}_{t,c}, B_c)$, where $\mathcal{I} \subseteq [N]$, $|\mathcal{I}| \le B_c$.
    - $*$ Send $(sg_{t,c,i})_{i \in \mathcal{SI}_c}$ to servers, where $sg_{t,c,i} \in \mathbb{Z}_q$ is the i-th entry of $\vec{\boldsymbol{g}}_{t,c}$ (static sparsity)
    - $*$ $\mathcal{DI} \leftarrow \mathcal{I} \setminus \mathcal{SI}_c$.
    - $*$ Send $\mathcal{G}_{t,c} \leftarrow \{(i, g_{t,c,i})\}_{i \in \mathcal{DI}}$ to the server (dynamic sparsity)
  - Server:
    - $*$ For $c \in \mathcal{AC}_t$:
      - $\cdot$ $\vec{\boldsymbol{g}}_{t,c} \leftarrow \mathsf{Vec}(\mathcal{G}_{t,c})$, *i.e.*, $g_{t,c,i} = m_i$ for $(i, m_i) \in \mathcal{G}_{t,c}$.
      - $\cdot$ $\vec{\boldsymbol{g}}_{t,c} \leftarrow \vec{\boldsymbol{g}}_{t,c} + sg_{t,c,i} \cdot \vec{\boldsymbol{e}}_i$ for $i \in \mathcal{SI}_c$, where $\vec{\boldsymbol{e}}_i$ is the basis vector.
    - $*$ $\vec{\boldsymbol{g}}_t \leftarrow (1/|\mathcal{C}_t|) \cdot \sum_{c=1}^{n_c} \vec{\boldsymbol{g}}_{t,c}$.
    - $*$ $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \vec{\boldsymbol{g}}_t$
    - $*$ Publish model $\boldsymbol{\theta}_{t+1}$.

Figure 1. Plaintext FL protocol to be secured

$e \bmod q$) where $\mathbf{a}_i, \mathbf{s} \in \mathbb{Z}_q^n$ are uniform and e comes from a B-bounded Gaussian, no efficient algorithm can distinguish them from uniformly random pairs $(\mathbf{a}_i, r_i)$.

Pseudorandom Functions. A PRF is a family of deterministic functions indexed by a key in $\mathcal{K}$ that map an input in $\mathcal{X}$ to an output in $\mathcal{Y}$ in such a way that the indexed function is computationally indistinguishable from a truly random function from $\mathcal{X}$ to $\mathcal{Y}$. In practice, PRFs are often instantiated using efficient block ciphers such as AES.

Definition 1 (**Distributed Point Functions** [24], [26]**).** An m-party DPF scheme is a pair of efficient algorithms $(\mathsf{DPF.KeyGen}, \mathsf{DPF.Eval})$ for a point function $f_{\alpha,\beta}\colon [N] \to \mathbb{G}$, which outputs $\beta \in \mathbb{G}$ if and only if $x = \alpha$, 0 otherwise.

- $\mathsf{KeyGen}(1^\lambda, \alpha, \beta)$: (a PPT algorithm) On the security parameter 1^λ, point $\alpha \in [N]$, and value $\beta \in \mathbb{G}$, the key generation algorithm outputs m keys, $\{k_j\}_{j \in [m]}$.
- $\mathsf{Eval}(k_j, x)$: (an efficient deterministic algorithm) On key k_j and input $x \in [N]$, outputs $y \in \mathbb{G}$. For convenience, we also define $\mathsf{EvalAll}(k_j)$ that outputs $\boldsymbol{y} \in \mathbb{G}^N$ with $y_i \leftarrow \mathsf{Eval}(k_j, i)$ for $i \in [N]$.

DPF should satisfy the following properties:

- **Correctness:** Given keys $\{k_j\}_{j \in [m]}$ for a point function $f_{\alpha,\beta}$, it holds that $\sum_{j \in [m]} \mathsf{Eval}(k_j, x) = f_{\alpha,\beta}(x)$.
- **Security:** Given a strict subset of $\{k_j\}_{j \in [m]}$ for a point function $f_{\alpha,\beta}$, an efficient adversary gains no information about α, β.

A concretely efficient 2-party DPF can be constructed with key size $O(\lambda \log N)$ [25]. For $m > 2$, a concretely efficient m-party DPF can be constructed with key size $O_\lambda(2^{m/2}\sqrt{N})$ [26] or $O_\lambda(m^4 \cdot \sqrt{N})$ [34]. Cryptographic details, *e.g.*, for constructions, are deferred to Appendix A.

4. Distributed Sparse PRFs

We introduce distributed *sparse* pseudorandom functions (DSPRFs), a new primitive central to Section 5. Classical distributed PRFs [35], [36], [37] use short keys to distribute additive shares of arbitrarily many random values. DSPRFs extend this capability with a private coordinate, enabling succinct placement of randomness at hidden indices.

4.1. Static Distributed Sparse PRFs

In the static variant, fix a single hidden core dimension $i^* \in [N]$ (for exposition, assume $i^* = 1$). Using short distributed keys, the parties can generate, for arbitrarily many queries, additive shares of independent "unit" *sparse random* vectors supported on i^*: for each query, the vector $\vec{\boldsymbol{v}}$ satisfies $\vec{\boldsymbol{v}}[i^*] = r$ for a random r (over the value domain), and $\vec{\boldsymbol{v}}[j] = 0$ for all $j \neq i^*$. Equivalently, the parties obtain additive shares of independent samples $\vec{\boldsymbol{v}}^{(1)}, \vec{\boldsymbol{v}}^{(2)}, \dots$, each supported at i^* with a single nonzero entry.

Definition 2 (Static Distributed Sparse PRFs (StaF)**).** Let $\mathcal{X}$ be the input space. Let $\mathcal{R}$ be an integer ring that serves as the output space. Let $N, m \in \mathbb{N}$ be integers. An N-dimensional m-party StaF is a tuple of algorithms $(\mathsf{KeyGen}, \mathsf{CoreEval}, \mathsf{DistEval})$.

- $\mathsf{KeyGen}(1^\lambda, i^*)$: (a PPT algorithm) On input 1^λ, and core (private) dimension $i^* \in [N]$, output a master key mk, and distributed keys $(dk_j)_{j \in [m]}$.
- $\mathsf{CoreEval}(mk, x)$: (an efficient deterministic algorithm) On master key mk, and $x \in \mathcal{X}$, output a value $y \in \mathcal{R}$.
- $\mathsf{DistEval}(dk_j, x)$: (an efficient deterministic algorithm) On input distributed key dk_j, and $x \in \mathcal{X}$, output $(\vec{\boldsymbol{p}}_j, \vec{\boldsymbol{y}}_j) \in \mathcal{R}^N \times \mathcal{R}^N$.

(KeyGen, CoreEval, DistEval) is a static distributed sparse PRFs (StaF) if it satisfies:

1) **Correctness**: For any core dimension $i^* \in [N]$, let the generated key be $(mk, (dk_j)_{j\in[m]}) \leftarrow \mathsf{KeyGen}(1^\lambda, i^*)$, for any $x \in \mathcal{X}$, let $y \leftarrow \mathsf{CoreEval}(mk, x)$, $(\vec{\boldsymbol{p}}_j, \vec{\boldsymbol{y}}_j) \leftarrow \mathsf{DistEval}(dk_j, x)$, then it holds that
$$\sum_{j\in[m]} \vec{\boldsymbol{p}}_j = \vec{\boldsymbol{e}}_{i^*}, \text{ and } \sum_{j\in[m]} \vec{\boldsymbol{y}}_j = y \cdot \vec{\boldsymbol{e}}_{i^*},$$
where $\vec{\boldsymbol{e}}_{i^*} \in \mathcal{R}^N$ is the basis vector, *i.e.*, equal to $1 \in \mathcal{R}$ at index i^* and 0 elsewhere.
2) **Security**: Given a strict subset of distributed keys $\{dk_j\}_{j\in[m]}$, the core dimension i^* remains hidden, and $F_{mk}(x) := \mathsf{CoreEval}(mk, x)$ remains a pseudorandom function (over output space $\mathcal{R}$).

Compared to Distributed PRFs. Static DSPRFs generalize distributed PRFs in that they can distribute pseudorandom values *placed at a hidden coordinate* $i^* \in [N]$. Setting $N = 1$ (which forces $i^* = 1$) recovers distributed PRFs.

Compared to Distributed Point Functions (DPFs). Static DSPRFs generalize DPFs [24] in that they can distribute *many* additive shares of one sparse vector, whereas a DPF yields only one additive share.

We then define an approximate StaF by relaxing the correctness property to allow a small additive error. This relaxed version suffices for our application.

B-Approximate Correctness. For any core dimension $i^* \in [N]$, let the key be $(mk, (dk_j)_{j\in[m]}) \leftarrow \mathsf{KeyGen}(1^\lambda, i^*)$, for any $x \in \mathcal{X}$, let $y \leftarrow \mathsf{CoreEval}(mk, x)$, $(\vec{\boldsymbol{p}}_j, \vec{\boldsymbol{y}}_j) \leftarrow \mathsf{DistEval}(dk_j, x)$, there exists $\gamma \in [-B, B]$ such that
$$\sum_{j\in[m]} \vec{\boldsymbol{p}}_j = \vec{\boldsymbol{e}}_{i^*}, \text{ and } \sum_{j\in[m]} \vec{\boldsymbol{y}}_j = (y+\gamma) \cdot \vec{\boldsymbol{e}}_{i^*}$$

Figure 2 formally describes our approximate StaF construction. Readers may skip this construction without losing continuity in understanding the remainder of this paper.

Theorem 1. *Figure 2 presents a B-approximate static distributed sparse PRF (Definition 2) if LWE is hard.*

Proof. Security follows from the DPF that one can learn nothing from any subset of the distributed keys. CoreEval is a pseudorandom function based on the LWE assumption. Correctness is proven in Appendix C.1. □

Construction of StaF

Public parameter:
- Dimension N, number of parties m.
- LWE parameter: (n, q, B), let $\mathcal{R} = \mathbb{Z}_q^n$.
- Input domain $\mathcal{X} = \{0,1\}^\lambda$.

Building blocks:
- m-party DPF (Definition 1) scheme: $[N] \rightarrow \mathcal{R}$, with key size $s(m, N, |\mathcal{R}|)$
- Hash function $H_1 \colon \mathcal{X} \rightarrow \mathbb{Z}_q^n$.
- Hash function $H_2 \colon \mathcal{X} \times \mathcal{R} \rightarrow \{0,1\}^*$
- B-bounded Gaussian error sampling function $\mathsf{Gau} \colon \{0,1\}^* \times \mathbb{Z} \rightarrow \mathbb{Z}_q$

Construction:
- $\mathsf{KeyGen}(1^\lambda, i \in [N])$:
  - Sample $mk \leftarrow \mathcal{R}$.
  - $(k_j)_{j\in[m]} \leftarrow \mathsf{DPF.KeyGen}(1^\lambda, i, mk)$.
  - $(k'_j)_{j\in[m]} \leftarrow \mathsf{DPF.KeyGen}(1^\lambda, i, 1)$.
  - $dk_j \leftarrow (k_j, k'_j)$ for $j \in [m]$.
  - Output $(mk, (dk_j)_{j\in[m]})$.
- $\mathsf{CoreEval}(mk, x \in \mathcal{X})$:
  - $e \leftarrow \mathsf{Gau}(H_2(x||mk), B) \in \mathbb{Z}_q$.
  - Parse $mk = \boldsymbol{s} \in \mathbb{Z}_q^n$.
  - $y \leftarrow \langle H_1(x), \boldsymbol{s}\rangle + e \in \mathbb{Z}_q$.
  - Output $y \in \mathbb{Z}_q$.
- $\mathsf{DistEval}(dk_j, x \in \mathcal{X})$:
  - Parse $dk_j = (k_j, k'_j)$.
  - $\boldsymbol{S}_j \leftarrow \mathsf{DPF.EvalAll}(k_j) \in \mathbb{Z}_q^{n\times N}$
  - $\vec{\boldsymbol{p}}_j \leftarrow \mathsf{DPF.EvalAll}(k'_j) \in \mathbb{Z}_q^N$
  - $\vec{\boldsymbol{y}}_j \leftarrow \langle H_1(x), \boldsymbol{S}_j\rangle \in \mathbb{Z}_q^N$.
  - Output $(\vec{\boldsymbol{p}}_j, \vec{\boldsymbol{y}}_j) \in \mathbb{Z}_q^N \times \mathbb{Z}_q^N$.

Efficiency:
- Master key size: $\mathsf{size}(mk) = n \log q$.
- Distributed key size: $\mathsf{size}(k'_j) = 2 \cdot s(m, N, |\mathcal{R}|)$.

Figure 2. Our (approximate) static distributed sparse PRFs

4.2. Dynamic Distributed Sparse PRFs

A static distributed sparse PRF that fixes a single hidden coordinate i^* can be extended to a set of core dimensions $\mathcal{I}^* \subseteq [N]$ by instantiating the scheme for each $i^* \in \mathcal{I}^*$ and summing the resulting vectors. This still requires that the hidden set $\mathcal{I}^*$ be fixed and known to the client at KeyGen.

Our ideal goal is to distribute short keys that enable the parties to obtain additive shares of an *unbounded* sequence of unit sparse random vectors whose support coordinate is chosen at *random* per query. For a fresh input x, $\mathsf{CoreEval}(mk, x)$ outputs both a random position $p \in [n]$ and a random value y. The parties derive additive shares of the vector $\vec{\boldsymbol{v}}$ with $\vec{\boldsymbol{v}}[p] = y$ and $\vec{\boldsymbol{v}}[j] = 0$ for all $j \neq p$. To obtain a vector supported at a specific position p, the client may use rejection sampling: repeatedly invoke $\mathsf{CoreEval}(mk, x)$ on fresh inputs x until the returned position equals p. Crucially, without the master key, observing x reveals nothing about the associated position p or value y.

Definition 3 (Dynamic Distributed Sparse PRFs (DynF))**.** Let $\mathcal{X}$ be the input space. Let $\mathcal{R}$ be an integer ring that serves as the output space. Let $N, m \in \mathbb{N}$ be integers. An N-dimensional m-party DynF consists of the following algorithms.

- $\mathsf{KeyGen}(1^\lambda)$: (a PPT algorithm) On the security parameter 1^λ, output a master key $mk \in \mathcal{K}$, and distributed

Construction of DynF

Public parameter:

- Dimension N, number of parties m.
- LWE parameter: (n, q, B), let $\mathcal{R} = \mathbb{Z}_q^n$.
- Input domain $\mathcal{X}$ with $|\mathcal{X}| = \mathsf{poly}(\lambda)$.

Building blocks:

- Degree-d ($d \leq 5$) $\mathsf{RSPRG}\colon \Sigma_{\mathsf{in}}^k \to [N]^{k^d}$.
- m-party HSS (Definition 7) for degree-d ($d \leq 5$) computation over $\mathcal{R}$.
- Hash function $H_1\colon \mathcal{X} \to \mathbb{Z}_q^n$.
- Hash function $H_2\colon \mathcal{X} \times \mathbb{Z}_q^n \to \{0,1\}^*$
- B-bounded Gaussian error sampling function $\mathsf{Gau}\colon \{0,1\}^* \times \mathbb{Z} \to \mathbb{Z}_q$

Construction:

- $\mathsf{KeyGen}(1^\lambda)$:
  - Sample $\boldsymbol{s} \leftarrow \mathcal{R}$.
  - $(sd, \mathsf{ssd}) \leftarrow \mathsf{RSPRG.KeyGen}(1^\lambda, \boldsymbol{s})$, where $sd \in \Sigma_{\mathsf{in}}^k$ ($k = |\mathcal{X}|^{1/d}$), $\mathsf{ssd} \in \mathcal{R}^{2k'}$ ($k' = k|\Sigma_{\mathsf{in}}|$).
  - $(dk_1, \ldots, dk_m) \leftarrow \mathsf{HSS.Enc}(1^\lambda, \mathsf{ssd})$
  - $mk \leftarrow (sd, \boldsymbol{s})$
  - Output $mk, (dk_1, \ldots, dk_m)$.
- $\mathsf{CoreEval}(mk, x \in \mathcal{X})$:
  - Parse $mk = (sd, \boldsymbol{s})$, where $\boldsymbol{s} \in \mathbb{Z}_q^n$
  - $p \leftarrow \mathsf{RSPRG.CoreEval}(sd, x) \in [N]$.
  - $e \leftarrow \mathsf{Gau}(H_2(x \| mk), B) \in \mathbb{Z}_q$
  - $y \leftarrow \langle H_1(x), \boldsymbol{s} \rangle + e \in \mathbb{Z}_q$.
  - Output $(p, y) \in [N] \times \mathbb{Z}_q$.
- $\mathsf{DistEval}(dk_j, x \in \mathcal{X})$:
  - Compute $(\boldsymbol{a}_j, \boldsymbol{b}_j) \in \mathcal{R}^N \times \mathcal{R}^N$:
  $$(\boldsymbol{a}_j, \boldsymbol{b}_j) = \mathsf{HSS.DistEval}(dk_j, f(\cdot))$$
  where $f(\cdot) := \mathsf{RSPRG.SparseEval}(\cdot, x)$ is a multivariate polynomial of degree d.
  - Parse $\boldsymbol{a}_j = (\vec{\boldsymbol{p}}_j, \ldots)$ for $\vec{\boldsymbol{p}}_j \in \mathbb{Z}_q^N$.
  - Parse $\boldsymbol{b}_j = \boldsymbol{S}_j \in \mathbb{Z}_q^{n \times N}$.
  - $\vec{\boldsymbol{y}}_j \leftarrow \langle H_1(x), \boldsymbol{S}_j \rangle \in \mathbb{Z}_q^N$.
  - Output $(\vec{\boldsymbol{p}}_j, \vec{\boldsymbol{y}}_j) \in \mathbb{Z}_q^N \times \mathbb{Z}_q^N$.

Efficiency:

- Master key size:
$$\mathsf{size}(mk) = |\mathcal{X}|^{1/d} \cdot \mathsf{poly}(\lambda) + n \log q.$$
- Distributed key size:
$$\mathsf{size}(dk_j) = |\mathcal{X}|^{1/d} \cdot \log N \cdot \mathsf{poly}(\lambda).$$

Figure 3. Our (approximate) dynamic distributed sparse PRFs for a polynomial-size input domain with short keys

keys $(dk_j)_{j \in [m]}$.

- $\mathsf{CoreEval}(mk, x)$: (an efficient deterministic algorithm) On master key mk, input $x \in \mathcal{X}$, deterministically output $(p, y) \in [N] \times \mathcal{R}$.
- $\mathsf{DistEval}(dk_j, x)$: (an efficient deterministic algorithm) On distributed key dk_j, input $x \in \mathcal{X}$, output $(\vec{\boldsymbol{p}}_j, \vec{\boldsymbol{y}}_j) \in \mathcal{R}^N \times \mathcal{R}^N$.

DynF requires the following properties:

- **Correctness**: For any $x \in \mathcal{X}$, let $(mk, (dk_j)_{j \in [m]}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, $(p, y) \leftarrow \mathsf{CoreEval}(mk, x)$, $(\vec{\boldsymbol{p}}_j, \vec{\boldsymbol{y}}_j) \leftarrow \mathsf{DistEval}(dk_j, x)$, then it holds that
$$\sum_{j \in [m]} \vec{\boldsymbol{p}}_j = \vec{\boldsymbol{e}}_p, \text{ and } \sum_{j \in [m]} \vec{\boldsymbol{y}}_j = y \cdot \vec{\boldsymbol{e}}_p,$$
where $\vec{\boldsymbol{e}}_p \in \mathcal{R}^n$ is the basis vector, *i.e.*, equal to $1 \in \mathcal{R}$ at index p and 0 elsewhere.
- **Security**: Given a strict subset of distributed keys $\{dk_j\}_{j \in [m]}$, $F_{mk}(x) := \mathsf{CoreEval}(mk, x)$ remains a pseudorandom function (over output space $[n] \times \mathcal{R}$).

Compared to Pseudorandom Correlation Functions (PCFs). At a high level, dynamic DSPRFs are closely related to (the most general notion of) PCFs [38]. Here, we are specifically interested in the additive correlation of random sparse vectors. In addition, we explicitly require a $\mathsf{CoreEval}$ algorithm that gives the client more efficient access to the target correlation without evaluating $\mathsf{DistEval}$, which could be less efficient.

We then define an approximate DynF by relaxing the correctness property to allow a small additive error. This relaxed version will suffice for our application.

B-Approximate Correctness. For any core dimension $i^* \in [n]$, let the key be $(mk, (dk_j)_{j \in [m]}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, for any $x \in \mathcal{X}$, let $(p, y) \leftarrow \mathsf{CoreEval}(mk, x)$, $(\vec{\boldsymbol{p}}_j, \vec{\boldsymbol{y}}_j) \leftarrow \mathsf{DistEval}(dk_j, x)$, there exists $\gamma \in [-B, B]$ such that
$$\sum_{j \in [m]} \vec{\boldsymbol{p}}_j = \vec{\boldsymbol{e}}_p, \text{ and } \sum_{j \in [m]} \vec{\boldsymbol{y}}_j = (y + \gamma) \cdot \vec{\boldsymbol{e}}_p$$

Our construction of approximate DynF for input domain $\mathcal{X}$ of polynomial size is formally described in Figure 3, which is sufficient for our applications (see Section 5.2 on how to have "unbounded number of instances" from a bounded input domain; a small key size relative to the input domain is crucial). Readers may skip this construction without losing continuity in understanding the remainder of the paper. RSPRG denotes a low-degree random sparse PRG (Figure 13) and is reviewed in Appendix B.

Theorem 2. *Assuming LWE is hard, the construction in Figure 3 is a B-Approximate Dynamic Distributed Sparse PRFs (Definition 3) for a polynomial-size input domain.*

Proof. See Appendix C.2. □

5. Scalable Federated Learning

We present a scalable FL system built on distributed sparse PRFs. The protocol comprises:

- **Client Registration Subroutine** (Figure 14 in Appendix D): A client joins the system and distributes

Client in round t of FL

Global parameter: Model parameter θ_t.
Client public parameter:

- Bandwidth bound $B_t \leq B$.
- Selected servers $\mathcal{S}$ at registration, $|\mathcal{S}| = n'_s$.
- Counter cnt, counter bound M.

Client private parameter:

- Static sparse dimensions $\mathcal{SI} \subseteq [N]$.
- Encryption key EncKey (Figure 14).
- Dynamic key DynKey' (Figure 14).

Building blocks:

- Any machine-learning algorithm Train.
- Any gradient sparsification algorithm Sparse.
- Sparse gradient encryption subroutine SEnc (Figure 6).
- Registration subroutine $\mathsf{Register}$ (Figure 14).

On input: training data X_t.

1) **Status check:**
    - If not registered, run $\mathsf{Register}$ or skip this round.
    - If registered, choose to participate in the current round or not. If not, skip this round.
    - If not selected by the server, skip this round.
2) **Local training:**
    - $\vec{g} \leftarrow \mathsf{Train}(\theta_t, X_t) \in \mathbb{Z}_q^N$.
3) **Bandwidth-adaptive gradient sparsification:**
    - $\mathcal{I} \leftarrow \mathsf{Sparse}(\vec{g}, B_t)$, where $\mathcal{I} \subseteq [N]$, $|\mathcal{I}| \leq B_t$.
4) **Encrypt sparse gradient:**
    - $\mathcal{CT} \leftarrow \mathsf{SEnc}(\vec{g}, \mathsf{EncKey}, \mathcal{I})$.
5) **Broadcast encrypted gradient:**
    - Broadcast $\mathcal{CT}$ to all servers $\mathcal{S}$.
6) **Stream distributed key for DynF:**
    - Parse $\mathsf{DynKey}' = mk', \{dk'_j\}_{j \in \mathcal{S}}$.
    - $t \leftarrow \mathsf{cnt} - \mathsf{lstcnt}$.
    - Fetch next t bits of $\{dk'_j\}$, denoted by bitstr.
    - Send $\mathsf{bitstr} \in \{0,1\}^t$ to server j^*, where j^* is the part to which bitstr belongs.
    - $\mathsf{lstcnt} \leftarrow \mathsf{cnt}$.
    - If set $\{dk'_j\}$ has been fully streamed:
        - Reset counter $\mathsf{cnt} \leftarrow 0$.
        - Update $\mathsf{DynEncKey} \leftarrow mk'$.
        - Update $\mathsf{DynKey}' \leftarrow \mathsf{DynF.KeyGen}(1^\lambda, M)$.

Figure 4. Client in round t of FL: **Clients may be offline in any round; a dropout is equivalent to selecting "skip this round" in Step (1).** Selecting "skip this round" in Step (1) omits all Steps (2)–(6).

DSPRF keys to the servers. Once registered, the client can participate in any number of rounds.

- **Client Protocol in Round** t (Figure 4): Procedures executed by each (registered) client in round t.
- **Server Protocol in Round** t (Figure 5): Server-side operations in round t.
- **Client Sparse-Gradient Encryption** (Figure 6): Client-side subroutine to encrypt sparse gradients.
- **Server Partial Decryption** (Figure 7): Server-side subroutine to obtain additive shares for aggregation.

Server j in round t of FL

Parameter:
Server storage:

- $\mathcal{RC}_t \subseteq [n_c]$ is the set of registered clients.
- $\{\mathsf{PDecKey}_j^{(c)}\}_{c \in \mathcal{RC}_t}$.

Building blocks:

- Partial decryption subroutine $\mathsf{PDecrypt}$ (Figure 7).

On input:

- Registered and active set of clients $\mathcal{AC}_t \subseteq \mathcal{RC}_t$.
- Registration requests $\{\mathsf{PDecKey}_c\}_{c \in \mathcal{NC}_t}$, where $\mathcal{NC}_t$ is the new client set submitting such requests.

Execute:

1) **Status check:**
    - If $j = 1$ (the main server): Select a subset of clients $\mathcal{C}_t \leftarrow \mathsf{ClientSelect}(\mathcal{AC}_t)$ to participate in the current round.
    - For newly registered client $c \in \mathcal{NC}_t$:
        - Store registration information $\mathsf{PDecKey}_j^{(c)}$.
    - $\mathcal{RC}_{t+1} \leftarrow \mathcal{RC}_t \bigcup \mathcal{NC}_t$.
2) Initialize $\mathbf{sum}_j \leftarrow \mathbf{0} \in \mathbb{Z}_q^N$.
3) **Receive** $\mathcal{CT}^{(c)}$ **broadcasted by** $c \in \mathcal{C}_t$:
    - $\mathbf{sum}_j^{(c)} \leftarrow \mathsf{PDecrypt}(\mathcal{CT}^{(c)}, \mathsf{PDecKey}_j^{(c)})$.
    - $\mathbf{sum}_j \leftarrow \mathbf{sum}_j + \mathbf{sum}_j^{(c)}$.
4) **Receive** bitstr **sent by** $c \in \mathcal{C}_t$:
    - $\mathsf{bitstr}^{(c)} \leftarrow \mathsf{bitstr}^{(c)} || \mathsf{bitstr}$.
    - If stream completes, update.
5) **Aggregation:**
    - Send $\mathbf{sum}_j$ to server 1
    - If $j = 1$ (server 1):
        - Receive $\mathbf{sum}_j$ from server $j \in [n_s]$.
        - Compute $\mathbf{sum} \leftarrow \sum_{j \in [n_s]} \mathbf{sum}_j \in \mathbb{Z}_q^N$.
        - Update $\theta_{t+1} \leftarrow \theta_t - \mathbf{sum}/|\mathcal{C}_t|$.
        - Publish the updated model θ_{t+1}.

Figure 5. Server j in round t of FL

5.1. Encrypting Static Sparse Gradient

We begin with a static sparsity setting: the client has a fixed subset $\mathcal{SI} \subset [N]$ such that, in every training round, only coordinates in $\mathcal{SI}$ will be nonzero. This arises naturally in embedding layers of language models and recommendation systems, where $\mathcal{SI}$ corresponds to private tokens or purchased items. For simplicity, assume $\mathcal{SI} = \{i\}$.

In round $t \in [T]$, suppose that the client has a sparse gradient to upload, denoted by $\vec{g} \in \mathbb{Z}_q^N$, where $\vec{g}$ only has

```
Client sparse gradient encryption subroutine: SEnc
Building blocks:
  • $N$-dimensional StaF (Figure 2).
  • $N$-dimensional DynF (Figure 3).
On input:
  • Gradient $\vec{\boldsymbol{g}} \in \mathbb{Z}_q^N$.
  • Encryption key EncKey.
  • Selected index $\mathcal{I} \subseteq [N]$.
Execute:
  • Encrypt static sparse gradient:
    – Parse StaEncKey $= (mk_1, \ldots, mk_{|\mathcal{SI}|})$.
    – Initialize an empty list $\mathcal{SCT}$.
    – For each index $i \in \mathcal{SI}$:
      ∗ Encrypt gradient $g_i \in \mathbb{Z}_q$:
            $\mathsf{ctxt} \leftarrow g_i - \mathsf{StaF.CoreEval}(mk_i, t) \in \mathbb{Z}_q$
      ∗ $\mathcal{SCT}$.append(ctxt).
  • Encrypt dynamic sparse gradient:
    – Parse $mk =$ DynEncKey.
    – Initialize an empty list $\mathcal{DCT}$.
    – For each index $i \in \mathcal{I} \setminus \mathcal{SI}$:
      ∗ Increment counter cnt $\leftarrow$ cnt $+ 1$.
      ∗ Encrypt gradient $g_i \in \mathbb{Z}_q$:
            $\mathsf{ctxt}_1 \leftarrow p - i, \ \mathsf{ctxt}_2 \leftarrow g_i - y$
        where $(p, y) \leftarrow \mathsf{DynF.CoreEval}(mk, \mathsf{cnt}) \in [N] \times \mathbb{Z}_q$.
      ∗ ctxt $\leftarrow (\mathsf{ctxt}_1, \mathsf{ctxt}_2) \in [N] \times \mathbb{Z}_q$.
      ∗ $\mathcal{DCT}$.append(ctxt).
  • Output $\mathcal{CT} \leftarrow (\mathcal{SCT}, \mathcal{DCT})$.
```

Figure 6. Client sparse gradient encryption subroutine

a nonzero entry with value g_i at position i. So $\vec{\boldsymbol{g}}$ can be succinctly represented as $(i, g_i) \in [N] \times \mathbb{Z}_q$, where $\vec{\boldsymbol{g}} = g_i \cdot \vec{\boldsymbol{e}}_i$. In the plaintext version, the client can send i in the beginning, and only send $g_i \in \mathbb{Z}_q$ for each later round.

Recall that if the gradient is not sparse, then we can simply additively secret share the gradient and send each share to each server. Each server then computes the sum of shares of all clients, and then sums the results of the servers together. Correctness then follows with the linear homomorphism of additive secret sharing. The communication cost is similar to plaintext communication.

The challenge in extending the same kind of idea here is that if we additively secret share the value g_i and the position i (*i.e.*, $\vec{\boldsymbol{e}}_i$), then linear homomorphism does not allow the server to compute the product of two shares.

High-level idea. Our idea is to add a random mask r to the value g_i to obtain the ciphertext $c \in \mathbb{Z}_q$ and directly broadcast c. Suppose the servers hold secret shares of $\vec{\boldsymbol{e}}_i$; they can locally multiply those shares by c to obtain shares of $c \cdot \vec{\boldsymbol{e}}_i$. Now, the problem can be reduced to having the

```
Server $j$ partial decryption subroutine: PDecrypt
Building blocks:
  • $N$-dimensional StaF (Figure 2).
  • $N$-dimensional DynF (Figure 3).
On input:
  • Ciphertext $\mathcal{CT}^{(c)}$ of client $c$.
  • Partial decryption key $\mathsf{PDecKey}_j^{(c)}$ for client $c$.
Execute:
  • Parse $\mathcal{CT}^{(c)} = (\mathcal{SCT}^{(c)}, \mathcal{DCT}^{(c)})$.
  • Parse $\mathsf{PDecKey}_j^{(c)} = (\mathsf{StaDecKey}_j^{(c)}, \mathsf{DynDecKey}_j^{(c)})$.
  • Initialize $\mathbf{sum}_j^{(c)} \leftarrow \mathbf{0} \in \mathbb{Z}_q^N$.
  • Process the static sparse decrypting key:
    – If $\widetilde{\mathsf{StaDecKey}}_j^{(c)}$ is not cached,
      ∗ Initialize an empty list $\widetilde{\mathsf{StaDecKey}}_j^{(c)}$
      ∗ For each $dk_j$ in $\mathsf{StaDecKey}_j^{(c)}$:
        · $(\vec{\boldsymbol{p}}_j, \vec{\boldsymbol{y}}_j) \leftarrow \mathsf{StaF.DistEval}(dk_j, t) \in \mathbb{Z}_q^N$.
        · $\widetilde{\mathsf{StaDecKey}}_j^{(c)}$.append$((\vec{\boldsymbol{p}}_j, \vec{\boldsymbol{y}}_j))$.
  • Partially decrypt static sparsity part:
    – For (ctxt, $(\boldsymbol{p}_j, \boldsymbol{y}_j)$) in $(\mathcal{SCT}^{(c)}, \widetilde{\mathsf{StaDecKey}}_j^{(c)})$:
      ∗ Partially decrypt ctxt $\in \mathbb{Z}_q$ with key $\vec{\boldsymbol{p}}_j, \vec{\boldsymbol{y}}_j \in \mathbb{Z}_q^N$:
            $\vec{\boldsymbol{g}}_j^{(c)} \leftarrow \mathsf{ctxt} \cdot \vec{\boldsymbol{p}}_j + \vec{\boldsymbol{y}}_j \in \mathbb{Z}_q^N$
      ∗ $\mathbf{sum}_j^{(c)} \leftarrow \mathbf{sum}_j^{(c)} + \vec{\boldsymbol{g}}_j^{(c)} \in \mathbb{Z}_q^N$.
  • Partially decrypt dynamic sparsity part:
    – Parse $\mathsf{DynDecKey}_j^{(c)} = dk_j$.
    – For ctxt in $\mathcal{DCT}^{(c)}$:
      ∗ Parse ctxt $= (\mathsf{ctxt}_1, \mathsf{ctxt}_2) \in [N] \times \mathbb{Z}_q$.
      ∗ $(\vec{\boldsymbol{p}}_j, \vec{\boldsymbol{y}}_j) \leftarrow \mathsf{DynF.DistEval}(dk_j, \mathsf{cnt}) \in \mathbb{Z}_q^N \times \mathbb{Z}_q^N$.
      ∗ Partially decrypt ctxt:
            $\vec{\boldsymbol{g}}_j^{(c)} \leftarrow \mathsf{ctxt}_2 \cdot \mathsf{Shift}(\vec{\boldsymbol{p}}_j) + \mathsf{Shift}(\vec{\boldsymbol{y}}_j) \in \mathbb{Z}_q^N$
        where Shift $:=$ Shift$(\cdot, \mathsf{ctxt}_1)$ (Definition 4).
      ∗ $\mathbf{sum}_j^{(c)} \leftarrow \mathbf{sum}_j^{(c)} + \vec{\boldsymbol{g}}_j^{(c)} \in \mathbb{Z}_q^N$.
  • Output $\mathbf{sum}_j^{(c)} \in \mathbb{Z}_q^N$.
```

Figure 7. Server j partial decryption subroutine

servers compute the share of $r \cdot \vec{\boldsymbol{e}}_i$ in order to unmask $c \cdot \vec{\boldsymbol{e}}_i$. It seems like we are not making any progress because the servers still need to compute the product of two secret shares, which is the problem we initially encountered. However, one important observation is that since r can be any random value (its role is to mask g_i), our task is not jointly *calculating* the product of two *exact* private values, but jointly *running* a *probabilistic* process, *i.e.*, first sample a secret random value r, then put it into the secret dimension i. Our static distributed sparse PRFs (StaF) (Definition 2) exactly characterize what we essentially need here.

Our StaF-based solution. We first assume that the StaF has perfect correctness, and defer the analysis of the B-approximate StaF. In the beginning, the client generates a master key mk along with a set of distributed keys $\{dk_j\}$ for StaF tied to dimension i. It retains the master key mk locally and distributes dk_j to server j. In round t, let $\vec{g} = g_i \cdot \vec{e}_i \in \mathbb{Z}_q^N$ be the gradient. The client encrypts it by

$$c \leftarrow g_i - \mathsf{StaF.CoreEval}(mk, t) \in \mathbb{Z}_q.$$

Ciphertext c is only a scalar, not a vector. The client then directly publishes c. Server j can compute the additive share $\vec{g}_j \in \mathbb{Z}_q^N$ of the client's gradient $\vec{g}$ via:

$$\vec{g}_j \leftarrow c \cdot \vec{p}_j + \vec{y}_j, \text{ where}$$

$$(\vec{p}_j, \vec{y}_j) \leftarrow \mathsf{StaF.DistEval}(dk_j, t) \in \mathbb{Z}_q^N \times \mathbb{Z}_q^N$$

By the correctness of StaF, $\vec{p}_j$ and $\vec{y}_j$ are additive shares of $\vec{e}_i$ and $r \cdot \vec{e}_i$, respectively, where $r = \mathsf{StaF.CoreEval}(mk, t)$. The server j can now use the linear homomorphism of shares, locally add the vectors $\vec{g}_j$ for all clients, and send the result to server 1, which aggregates all such partial sums to obtain the final gradient.

Correctness under client dropouts. Correctness does not rely on universal participation in each round. The server evaluates DSPRFs only for active clients, so absent clients do not affect correctness (see Figure 5).

Efficiency. Compared to plaintext FL, clients have no online communication overhead.

The client computation overhead is one StaF.CoreEval invocation per gradient element, which boils down to one AES and one LWE inner product.

5.2. Encrypting Dynamic Sparse Gradient

Supporting dynamic sparsity patterns is more challenging. The client does not know which position will be sparse until the end of each training round, when the client obtains their sparsified gradients. The selected position will depend on the gradient after local training.

An initial unsuccessful attempt. To motivate our dynamic DSPRFs, we start with an attempt that tries to use static DSPRFs to deal with the dynamic sparse gradient. The (unsuccessful) attempt is to reduce the problem to the static case by letting $\mathcal{SI} = [N]$, where N is the total model dimension. In more detail, for each $i \in [N]$, the client distributes the distributed key dk_j tied to that i. However, this does not work for the following reasons:

- For each ciphertext of a gradient element (i, g_i), the client needs to tell the server which dk_j to use: tying dimension i to dk_j. Suppose that the client specifies the same dk_j across different rounds. Then, the server knows that there is an intersection of the sparse position between those rounds. Similarly, if the client specifies a different dk_j, the server knows it is a different position.

Our dynamic distributed PRFs (DynF) (Definition 3) exactly characterize what we essentially need here. Every evaluation yields a fresh pseudorandom index, so a server that lacks the master key learns nothing about the real coordinate it represents, and cannot correlate two outputs with the same underlying coordinate.

An initial approach based on DynF. Again, in the beginning, the client generates a master key mk along with a set of distributed keys $\{dk_j\}$ for DynF. It retains the master key mk locally and distributes dk_j to server j.

The client holding a gradient element (i, g_i) first keeps evaluating CoreEval on different input x:

$$(p, y) \leftarrow \mathsf{DynF.CoreEval}(mk, x)$$

until obtaining an output (p, y) such that $p = i$. Let the input be x'. The client encrypts g_i by

$$c \leftarrow g_i - y \in \mathbb{Z}_q.$$

Ciphertext c here is also only a scalar, not a vector. It then directly publishes $(c, x') \in \mathbb{Z}_q \times \mathcal{X}$. Server j can compute the additive share $\vec{g}_j$ of the client's gradient $\vec{g}$ via:

$$\vec{g}_j \leftarrow c \cdot \vec{p}_j + \vec{y}_j, \text{ where}$$

$$(\vec{p}_j, \vec{y}_j) \leftarrow \mathsf{DynF.DistEval}(dk_j, x') \in \mathbb{Z}_q^N \times \mathbb{Z}_q^N$$

Since CoreEval is pseudorandom, even though the server has the input x', it has no idea what index i it corresponds to by the security of DynF.

By the correctness of DynF, $\vec{p}_j$ and $\vec{y}_j$ are additive shares of $\vec{e}_i$ and $r \cdot \vec{e}_i$, respectively, where $r = \mathsf{DynF.CoreEval}(mk, t)$. The servers can then utilize the linear homomorphism to obtain the final sum.

Avoid rejection sampling. The above approach by rejection sampling will cause much computation overhead on the client and also ciphertext blowup (requiring sending $\log q + \lambda$ bits for $\mathcal{X} = \{0, 1\}^\lambda$, which could be roughly $2\times$ more communication than plaintext communication of $\log q + \log N$ bits). We now show how to get rid of rejection sampling. Suppose in round t, the client encrypting one gradient element (i, g_i) will evaluate CoreEval:

$$(p, y) \leftarrow \mathsf{DynF.CoreEval}(mk, t).$$

The client then encrypts (i, g_i) by

$$c_1 \leftarrow i - p, \text{ and } c_2 \leftarrow g_i - y.$$

Ciphertext $c = (c_1, c_2) \in [N] \times \mathbb{Z}_q$ consists of two scalars of total size $\log N + \log q$, the same as the plaintext. It then directly publishes c. Server j can compute the additive share $\vec{g}_j$ of the client's gradient $\vec{g}$ as follows:

$$\vec{g}_j \leftarrow c_2 \cdot \mathsf{Shift}(\vec{p}_j, c_1) + \mathsf{Shift}(\vec{y}_j, c_1)$$

where $\mathsf{Shift}\colon \mathbb{Z}_q^N \times [N] \to \mathbb{Z}_q^N$ is shifting the vector by the offset (formally defined in Definition 4), and

$$(\vec{p}_j, \vec{y}_j) \leftarrow \mathsf{DynF.DistEval}(dk_j, t) \in \mathbb{Z}_q^N \times \mathbb{Z}_q^N$$

Definition 4 (Shift Operation)**.** Let $\mathsf{Shift}\colon \mathbb{Z}_q^N \times [N] \to \mathbb{Z}_q^N$ be a shift operator. For any vector $\vec{\boldsymbol{v}} = (v_0, v_1, \ldots, v_{N-1}) \in \mathbb{Z}_q^N$ and any shift offset $r \in [N]$, define

$$\mathsf{Shift}(\vec{\boldsymbol{v}}, r)_i = v_{(i-r) \bmod N} \quad \text{for all } i \in [N].$$

That is, $\mathsf{Shift}(\vec{\boldsymbol{v}}, r)$ is the cyclic shift of $\vec{\boldsymbol{v}}$ by r positions.

It can be verified that $\mathsf{Shift}(\vec{\boldsymbol{p}}_j, c_1)$ and $\mathsf{Shift}(\vec{\boldsymbol{y}}_j, c_1)$ are shares of $\vec{\boldsymbol{e}}_i$ and $r \cdot \vec{\boldsymbol{e}}_i$, respectively, as required.

Supporting an unbounded number of rounds. Our construction of DynF (see Figure 3) supports only a polynomial-size input domain, say $|\mathcal{X}| = M$ for some $M = \mathsf{poly}(\lambda)$. Without loss of generality, assume that the client uploads exactly one gradient element per round. Then the key of DynF can only be used for up to M rounds of federated learning. Moreover, M must be determined in advance.

To overcome this limitation and support an unbounded number of rounds, we use a *pipelining* trick. The core idea is that each time the client uses an input $x \in \mathcal{X}$ to encrypt a gradient (*i.e.*, to evaluate the DynF), it simultaneously sends one bit of a new distributed key to the server.

The key property we exploit is that the size of each distributed key dk_j in our DynF construction grows only as $C \cdot M^{1/d}$, where C is independent of M. Therefore, if we choose M such that $C \cdot M^{1/d} \leq M$ (*i.e.*, $M \geq C^{d/(d-1)}$, resulting in a key size of $C^{d/(d-1)}$), then the client has enough rounds to fully transmit the next distributed key dk_{j+1} before exhausting the current input domain $\mathcal{X}$ of dk_j.

This pipelining mechanism allows the protocol to seamlessly switch to a fresh key once the current one is depleted, thereby supporting an unbounded number of rounds.

When clients skip the round. In the above analysis, we assume clients are available and participate in each round. When a client skips a round, it will not execute the key pipelining step (see Figure 4). This does not impact correctness or security: A skipped round does not count toward the key's usage (in the sense that the client does not evaluate the DynF on an unused input), so any client that skips the round need not transmit any part of the new keys. Therefore, when clients skip the round, correctness holds as in the static case in Section 5.1.

Efficiency. Compared to plaintext FL, clients incur only 1 bit of overhead per gradient element (to pipeline a new distributed key). The client computation overhead is one DynF.CoreEval invocation per gradient element, which boils down to one incremental local-PRG invocation (to yield the next output), one AES, and one LWE inner product.

5.3. Accuracy Analysis

Accuracy versus Sparsity. Per-entry gradient error is independent of sparsity. This is due to a key feature of our B-approximate distributed sparse PRFs: B-bounded Gaussian error appears only at the core dimension. Because each client's gradient elements lie in different dimensions, the errors introduced by a single client are distributed across disjoint coordinates and therefore do not accumulate. Moreover, since the final gradient is computed by averaging the gradients of many clients, each coordinate has at most a single small error per user, and the overall effect of noise is averaged out. As a result, the noise has a negligible impact on model training.

Theorem 3 (Approximate Correctness)**.** *Let the DSPRF be B-approximate correct. In round t, let $\vec{\boldsymbol{g}}_{(t)} = 1/|\mathcal{C}_t| \cdot \sum_{c\in\mathcal{C}_t} \vec{\boldsymbol{g}}_{t,c}$, i.e., a perfectly correct gradient, and $\vec{\boldsymbol{g}}'_{(t)} = 1/|\mathcal{C}_t| \cdot \mathsf{sum}^{(t)}$, i.e., the gradient output by our protocol with B-approximate distributed sparse PRFs, we have*

$$\left|\vec{\boldsymbol{g}}'_{(t)} - \vec{\boldsymbol{g}}_{(t)}\right| \leq \frac{B}{\sqrt{|\mathcal{C}_t|}}$$

5.4. Security Analysis

The pipelining trick is an implementation of DSPRF key distribution, so each server's view consists of the sum of gradients at each round, the ciphertexts of the participating clients at each round, and the distributed key(s) of each client. As long as not all servers collude, the distributed key(s) remain pseudorandom, so the ciphertext leaks nothing. Our protocol can be seamlessly integrated with differential privacy (DP). Specifically, each server independently adds appropriately calibrated noise to its local share of the summed gradients before final aggregation. We leave the details to the full version due to space constraints.

Theorem 4 (Security)**.** *Our protocol is secure in the presence of an adversary that corrupts any clients and any strict subset of servers.*

6. Experiment Evaluation

6.1. Experiment Setting

6.1.1. Dataset and Models. We evaluate our framework across the following three tasks.[3]

Image classification: For image classification tasks, we employ ResNet models [39] on three datasets: CIFAR-10, CIFAR-100, and Tiny ImageNet [40], [41]. To reduce communication cost, we employ top-k gradient sparsification and utilize dynamic DSPRFs to aggregate the gradients. For settings being non-independent and identically distributed (IID), we simulate by sampling from a Dirichlet distribution with $\beta = 1$. In each FL round, 100 clients are selected for gradient aggregation.

Language understanding: The language understanding tasks are evaluated on the RoBERTa model [42] using two General Language Understanding Evaluation

3. This work focuses on improving client communication and computation efficiency, and does not consider differential privacy, which is orthogonal to our contributions. Our accuracy experiments are intended to serve as an upper bound on the performance achievable under aggressive gradient sparsification. To ensure a clean comparison, we do not add DP noise any protocol. Standard DP mechanisms (*e.g.*, Gaussian noise) can be modularly integrated into all evaluated protocols, including ours.

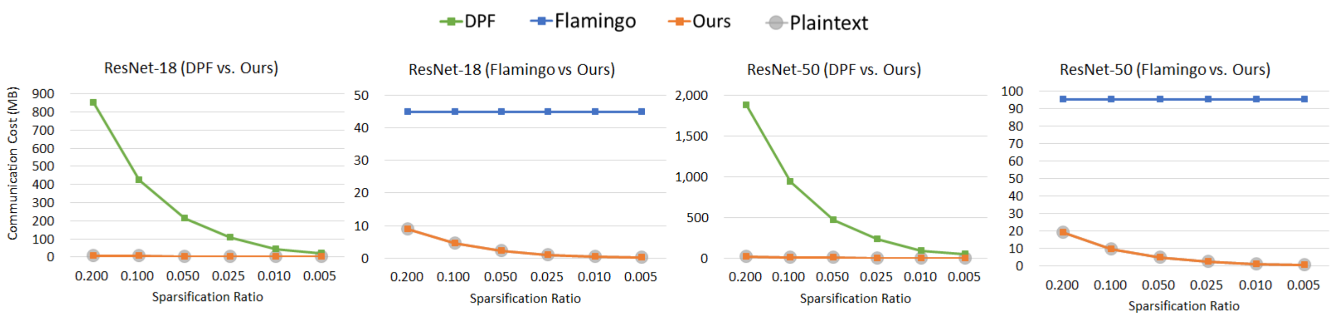

Figure 8. Per-client per-FL-round communication across sparsification ratios on CIFAR-100 (X-axis is on a logarithmic scale)

(GLUE) benchmarks: Corpus of Linguistic Acceptability (CoLA) [43] and Microsoft Research Paraphrase Corpus (MRPC) [44]. To improve finetuning efficiency, we employ low-rank adaptation (LoRA) [45] that introduces trainable rank-decomposition matrices into attention layers, and apply top-k gradient sparsification on the attention layers to reduce communication costs, while the token embedding layer is finetuned using our FL framework. We apply static and dynamic DSPRFs, respectively, on token embedding layers and attention layers. In each FL round, 20 clients are selected for gradient aggregation.

Recommendation systems: For the item embedding layer, we apply static DSPRFs using the indices of items previously purchased by the client. As only these embeddings receive nonzero gradients, this allows us to achieve communication savings. For the non-embedding layer, where gradients are generally dense and not predictable in advance, we apply the trivial static DSPRF over all dimensions. For the rating-prediction task, we evaluate our algorithm on four public datasets: MovieLens 1M (ML1M), MovieLens 10M (ML10M), MovieLens 25M (ML25M), and Yelp [46], [47]. For the Yelp dataset, we sample a portion of the top clients ranked in descending order by their number of rated items and obtain a subset containing $10,000$ clients and $93,386$ items. We consider two model-based recommender models: matrix factorization with biased term (MF) [48] and neural collaborative filtering (NCF) [49]. For the item ranking task, we evaluate our approach using two sequence models, Caser [50] and SASRec [51], on three categories of the Amazon datasets[4]: “Beauty”, “Sports”, and “Games”. For all datasets, we filter out clients with fewer than three ratings. For FL training, each user is treated as a client. In each round, 500 clients are chosen for ML1M, Yelp, and Amazon, 1000 clients for ML10M, and 2000 clients for ML25M.

6.1.2. LWE Parameters. In our experiments, we instantiate the LWE problem with modulus $q = 2^{32}$ (the same as the gradient modulus used in FL), dimension $n = 512$, and the error distribution chosen as a discrete Gaussian with standard deviation $\sigma = 3.2$, truncated to the interval $[-B, B]$ with $B = 20$. This B-bounded error model ensures a negligible truncation probability. Based on the LWE estimator [52] and current best known attacks, it achieves an estimated security level of at least 128 bits.

4. https://cseweb.ucsd.edu/~jmcauley/datasets/amazon/links.html

6.1.3. Baseline Methods. We evaluate our protocol against several aggregation protocols and message compression techniques, including: (1) Plaintext FL protocol that transmitted the nonzero elements in plaintext, (2) Flamingo [18] protocol, a recent state-of-the-art single-server multi-round secure aggregation method specifically for FL, and (3) a 2-party DPF approach that improves communication efficiency for sparse vectors [23], [25], [53], (4) 8-bit quantization (Bit8Quant) [7], and (5) ternary quantization (TernQuant) [8]. Bit8Quant and TernQuant are lossy message compression mechanisms applied to transmitted values. The baseline DPF approach can be optimized for static sparsity in embedding-based models by applying batching techniques. We incorporate these optimization strategies to reduce the communication cost of DPF. Without such optimizations, a naïve application of DPF would incur significantly higher communication overhead.

6.2. Client Communication

We compare the client communication of our protocol with plaintext FL, Flamingo, and DPF-based approach.

Image classification. The image classification task involves aggregating dynamic sparse gradients. We analyze communication overhead under varying sparsification ratios using ResNet-18 and ResNet-15 (Figure 8). Our protocol achieves communication costs comparable to plaintext FL. Compared to Flamingo, it reduces communication by $5\times$ to $200\times$ for sparsification ratios from 0.2 to 0.005 and achieves an average reduction of over $90\times$ compared with the 2-server DPF approach. Notably, DPF incurs a higher overhead than Flamingo when $\rho \geq 0.025$, while our protocol consistently outperforms Flamingo across different ρ.

Language understanding. The language understanding task includes the aggregation of both dynamic (attention layer) and static (token embedding layer) sparse gradients. Table 3 presents the communication cost under language models of different scales. Our protocol has almost the same overhead as the plaintext protocol, and the cost can be further optimized by applying sparsification on the embedding layer.

For the DPF approach, we apply 2-server DPF to the embedding layer (with batching optimization) and apply Flamingo to non-embedding layers. Communication savings are more pronounced at higher sparsity, with up to a $6\times$ reduction over DPF and a $600\times$ reduction over Flamingo at $\rho = 0.1$.

Recommendation systems. For the item-embedding layer, the communication cost is optimized with static distributed sparse PRFs. As Tables 5 and 6 show, our online communication cost matches the plaintext protocol, and the advantage grows with sparsity, reaching an average $830\times$ reduction on Yelp at 0.11% density against Flamingo. Compared to DPF, our method remains cheaper, which is about $4\times$ for rating prediction and $2.5\times$ for item ranking.

6.3. Client Computation Overhead

We compare the extra client computation cost from distributed sparse PRFs with the original training cost.

Static sparsity. For gradients with m nonzero elements, each client performs m AES and LWE operations. As shown in Figures 9 and 10, our algorithm adds only 0.87% overhead on average for recommendation tasks (rating prediction/item ranking) and 0.29% for language understanding versus baseline training.

Dynamic sparsity. Beyond static operations, dynamic sparsity additionally requires m local-PRG evaluations per client. While more costly than static sparsity, the extra computation overhead remains low compared with the original training cost, especially under a smaller sparsification ratio ρ. At $\rho \leq 0.1$, the extra cost is $\leq$8.9% of the training cost for CIFAR-100 (image classification) and $\leq$1.4% for CoLA (language understanding).

6.4. Utility Analysis

In our experiments, we apply a uniform sparsification ratio ρ for all clients. This setup serves as a lower bound on the achievable utility at each sparsification level, as in practice, clients could adapt their sparsification ratios based on their available bandwidth, and some clients may have sufficient bandwidth to upload their full gradients.

Image classification. We evaluate accuracy for top-k sparsification ratios $\rho \in [0.05, 0.2]$ and full gradients with $\rho = 1$ in Table 4. At $\rho = 0.2$, the average accuracy loss is about 0.7%, while communication drops by $5\times$ relative to full gradients. Smaller ρ yields larger savings but lower accuracy, enabling a tunable tradeoff between efficiency and utility.

Language understanding. We evaluate the accuracy (ACC), area under the curve (AUC), and overhead of our algorithm against two quantization-based message compression methods, Bit8Quant and TernQuant, in Table 7. Quantization yields only modest savings. Our protocol reduces communication by about $13\times$ relative to Bit8Quant and $26\times$ relative to TernQuant. These lossy compressors also degrade utility by 0.1%–3.8%. Appendix E.2 analyzes utility under varying sparsification ratios.

Recommendation systems. Tables 8 and 9 report utility and communication for rating prediction and item ranking. All methods aim to reduce the communication cost of the item-embedding layer, which dominates model size. By exploiting gradient sparsity, our protocol reduces communication substantially more than both quantization baselines across tasks while maintaining accuracy. In contrast, the lossy quantization methods Bit8Quant and TernQuant incur utility drops of up to 4.3% and 4.9%, respectively.

6.5. Server Computation

We optimize client communication and computation and leave server computation unoptimized because it is reasonably efficient. To decrypt dynamic sparse gradients, the server performs AES and LWE operations and runs an HSS evaluation with degree $d \leq 5$. Low-degree HSS has been shown to be concretely efficient [54]. Using those HSS measurements [54], we estimate that on a 2.6 GHz machine the server-side time per FL round is about $M/3$ seconds per core, where M is the total number of gradient elements from participating clients in that round. Further implementation optimizations may improve this empirical performance.

7. Conclusion

Cryptography-ML co-design has advanced private inference for neural networks [55] and transformers [56]. This paper revisits the long-standing tension between communication efficiency and privacy in federated learning. By co-designing cryptographic primitives with machine-learning optimizations, we introduce distributed sparse pseudorandom functions and show how they enable a communication-efficient FL protocol that enjoys the communication savings from sparsity in federated learning.

Our method requires multiple servers and assumes not all servers collude. We leave it as future work to extend our protocol to the malicious setting.

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” in *AISTATS*, 2017, pp. 1273–1282.

[2] European Parliament and Council of the European Union, “Regulation (EU) 2016/679 of the european parliament and of the council,” *General Data Protection Regulation (GDPR), Official Journal of the European Union*, vol. 59, no. L119, 2016.

[3] Z. Xu, Y. Zhang, G. Andrew, C. A. Choquette-Choo, P. Kairouz, H. B. McMahan, J. Rosenstock, and Y. Zhang, “Federated learning of Gboard language models with differential privacy,” in *ACL*, 2023, pp. 629–639.

[4] L. Zhu, Z. Liu, and S. Han, “Deep leakage from gradients,” in *NeurIPS*, 2019, pp. 14 747–14 756.

[5] D. I. Dimitrov, M. Balunovic, N. Konstantinov, and M. T. Vechev, “Data leakage in federated averaging,” *Trans. Mach. Learn. Res.*, vol. 2022, 2022.

[6] D. I. Dimitrov, M. Baader, M. N. Müller, and M. T. Vechev, “SPEAR: exact gradient inversion of batches in federated learning,” in *NeurIPS*, 2024.

TABLE 3. Per-client communication cost (in MB) in one FL round for language understanding tasks on the CoLA dataset under static sparsity of embedding layers and varying (dynamic) top-k sparsification ratio ρ on non-embedding layers (The (optimized) DPF only applies to the static embedding layer, so its cost does not change with ρ.)

		DistallBert (67M)	Roberta-Large (355M)	Qwen2.5 (1.5B)	Llama-3.1 (8B)	Llama-3.3 (70B)
$\rho = 0.1$	Plaintext	0.83	1.49	1.56	4.35	12.56
	Flamingo	96.72	213.24	937.87	2115.01	4268.29
	DPF	4.10	8.95	6.71	19.74	77.70
	Ours	0.83	1.49	1.56	4.35	12.56
$\rho = 0.3$	Insecure	1.42	2.96	2.43	7.09	25.68
	Flamingo	96.72	213.24	937.87	2115.01	4268.29
	DPF	4.10	8.95	6.71	19.74	77.70
	Ours	1.42	2.96	2.43	7.09	25.68
$\rho = 0.5$	Insecure	2.01	4.43	3.31	9.82	38.80
	Flamingo	96.72	213.24	937.87	2115.01	4268.29
	DPF	4.10	8.95	6.71	19.74	77.70
	Ours	2.01	4.43	3.31	9.82	38.80

TABLE 4. Accuracy on image classification under varying top-k sparsification ratio ρ

ρ	1	0.20	0.15	0.10	0.05
CIFAR-10	$0.905_{\pm 0.002}$	$0.904_{\pm 0.007}$	$0.904_{\pm 0.008}$	$0.900_{\pm 0.002}$	$0.901_{\pm 0.003}$
CIFAR-100	$0.634_{\pm 0.000}$	$0.630_{\pm 0.001}$	$0.625_{\pm 0.002}$	$0.610_{\pm 0.000}$	$0.580_{\pm 0.003}$
Tiny ImageNet	$0.528_{\pm 0.001}$	$0.521_{\pm 0.000}$	$0.513_{\pm 0.001}$	$0.494_{\pm 0.001}$	$0.474_{\pm 0.002}$

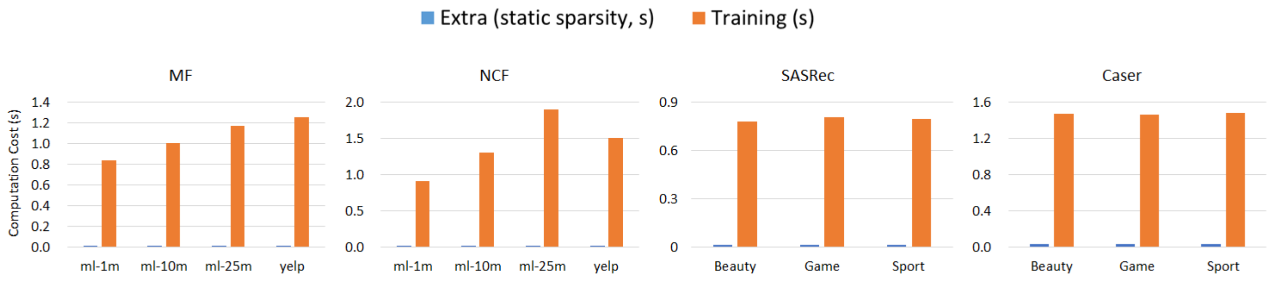

Figure 9. Client computation cost (s) (per FL round) for rating prediction and item ranking tasks

TABLE 5. Client communication (MB) per round for rating prediction task

	Method	ML1M	ML10M	ML25M	Yelp
	Size m	$3,883$	$10,681$	$62,423$	$93,386$
	Density	4.26%	1.34%	0.25%	0.11%
MF	Plaintext	0.04	0.04	0.04	0.03
	Flamingo	0.99	2.73	15.98	23.91
	DPF	0.15	0.14	0.16	0.11
	Ours	0.04	0.04	0.04	0.03
NCF	Plaintext	0.02	0.03	0.04	0.02
	Flamingo	0.50	2.06	11.99	14.95
	DPF	0.11	0.12	0.14	0.09
	Ours	0.02	0.03	0.04	0.02

TABLE 6. Client communication (Comm., MB) per round for item ranking task

		Beauty	Game	Sport
Caser	Plaintext	0.41	0.42	0.41
	Flamingo	44.26	57.85	83.93
	DPF	1.02	1.05	1.02
	Ours	0.41	0.42	0.41
SASRec	Plaintext	0.27	0.27	0.27
	Flamingo	44.32	57.91	83.99
	DPF	0.66	0.68	0.67
	Ours	0.27	0.27	0.27

[7] T. Dettmers, "8-bit approximations for parallelism in deep learning," in *ICLR (Poster)*, 2016.

[8] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "TernGrad: Ternary gradients to reduce communication in distributed deep learning," in *NIPS*, 2017, pp. 1509–1519.

[9] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," in *EMNLP*, 2017, pp. 440–445.

[10] J. Wangni, J. Wang, J. Liu, and T. Zhang, "Gradient sparsification

TABLE 7. Utility and online per-client communication cost (Comm.) in MB (per FL round) for our protocol and various message compression methods on RoBERTa in language understanding tasks at top-k sparsification ratio $\rho = 0.5$

	ACC	AUC	Comm.
		CoLA	
Bit8Quant	$0.840_{\pm 0.004}$	$0.815_{\pm 0.003}$	53.31
TernQuant	$0.839_{\pm 0.004}$	$0.815_{\pm 0.008}$	26.65
Ours	$\mathbf{0.841}_{\pm \mathbf{0.002}}$	$\mathbf{0.817}_{\pm \mathbf{0.003}}$	**2.01**
		MRPC	
Bit8Quant	$0.842_{\pm 0.005}$	$0.816_{\pm 0.006}$	53.31
TernQuant	$0.843_{\pm 0.003}$	$0.829_{\pm 0.009}$	26.65
Ours	$\mathbf{0.864}_{\pm \mathbf{0.012}}$	$\mathbf{0.849}_{\pm \mathbf{0.008}}$	**2.05**

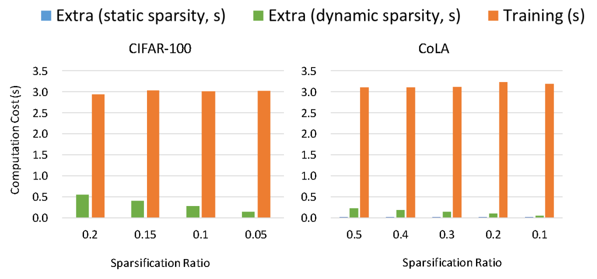

Figure 10. Client computation cost (s) for language understanding and image classification w.r.t. sparsification ratios

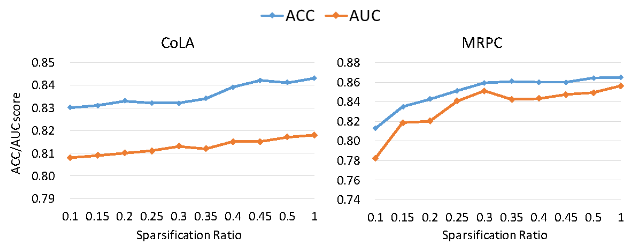

Figure 11. Utility on CoLA and MRPC with RoBERTa w.r.t. top-k sparsification ratios

for communication-efficient distributed optimization," in *NIPS*, 2018, pp. 1306–1316.

[11] K. A. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *CCS*, 2017, pp. 1175–1191.

[12] T. S. Nguyen, T. Lepoint, and N. Trieu, "Mario: Multi-round multiple-aggregator secure aggregation with robustness against malicious actors," in *EuroS&P*, 2025, pp. 1140–1164.

[13] R. Behnia, A. Riasi, R. Ebrahimi, S. S. M. Chow, B. Padmanabhan, and T. Hoang, "Efficient secure aggregation for privacy-preserving federated machine learning," in *ACSAC*, 2024, pp. 778–793.

[14] Y. Guo, A. Polychroniadou, E. Shi, D. Byrd, and T. Balch, "MicroSecAgg: Streamlined single-server secure aggregation," *Proc. Priv. Enhancing Technol.*, vol. 2024, no. 3, pp. 246–275, 2024.

[15] H. Li, H. Lin, A. Polychroniadou, and S. Tessaro, "LERNA: Secure single-server aggregation via key-homomorphic masking," in *ASIACRYPT Part I*, 2023, pp. 302–334.

[16] M. Rathee, C. Shen, S. Wagh, and R. A. Popa, "ELSA: Secure aggregation for federated learning with malicious actors," in *S&P*, 2023, pp. 1961–1979.

[17] Z. Liu, S. Chen, J. Ye, J. Fan, H. Li, and X. Li, "SASH: Efficient secure aggregation based on SHPRG for federated learning," in *UAI*, 2022, pp. 1243–1252.

[18] Y. Ma, J. Woods, S. Angel, A. Polychroniadou, and T. Rabin, "Flamingo: Multi-round single-server secure aggregation with applications to private federated learning," in *S&P*, 2023, pp. 477–496.

[19] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, "Secure single-server aggregation with (poly)logarithmic overhead," in *CCS*, 2020, pp. 1253–1269.

[20] J. So, B. Güler, and A. S. Avestimehr, "Turbo-Aggregate: Breaking the quadratic aggregation barrier in secure federated learning," *IEEE J. Sel. Areas Inf. Theory*, vol. 2, no. 1, pp. 479–489, 2021.

[21] J. Ma, S. Naas, S. Sigg, and X. Lyu, "Privacy-preserving federated learning based on multi-key homomorphic encryption," *Int. J. Intell. Syst.*, vol. 37, no. 9, pp. 5880–5901, 2022.

[22] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, robust, and scalable computation of aggregate statistics," in *NSDI*, 2017, pp. 259–282.

[23] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai, "Lightweight techniques for private heavy hitters," in *S&P*, 2021, pp. 762–776.

[24] N. Gilboa and Y. Ishai, "Distributed point functions and their applications," in *EUROCRYPT*, 2014, pp. 640–658.

[25] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing: Improvements and extensions," in *CCS*, 2016, pp. 1292–1303.

[26] ——, "Function secret sharing," in *EUROCRYPT Part II*, 2015, pp. 337–367.

[27] J. P. K. Ma, R. K. H. Tai, Y. Zhao, and S. S. M. Chow, "Let's stride blindfolded in a forest: Sublinear multi-client decision trees evaluation," in *NDSS*, 2021.

[28] L. K. L. Ng, S. S. M. Chow, A. P. Y. Woo, D. P. H. Wong, and Y. Zhao, "Goten: GPU-Outsourcing trusted execution of neural network training," in *AAAI*, 2021, pp. 14 876–14 883.

[29] M. Cong, S. S. M. Chow, S.-M. Yiu, and T. H. Yuen, "Scalable zk-SNARKs for matrix computation: A generic framework for verifiable deep learning," in *ASIACRYPT*, 2025, to appear.

[30] L. K. L. Ng and S. S. M. Chow, "SoK: Cryptographic neural-network computation," in *S&P*, 2023, pp. 497–514.

[31] S. U. Stich, J. Cordonnier, and M. Jaggi, "Sparsified SGD with memory," in *NeurIPS*, 2018, pp. 4452–4463.

[32] A. N. Sahu, A. Dutta, A. M. Abdelmoniem, T. Banerjee, M. Canini, and P. Kalnis, "Rethinking gradient sparsification as total error minimization," in *NIPS*, 2021, pp. 8133–8146.

[33] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," in *STOC*, 2005, pp. 84–93.

[34] A. Goel, M. Wang, and Z. Wang, "Multiparty distributed point functions," in *CRYPTO Part IV*, 2025, pp. 140–173.

[35] S. Micali and R. Sidney, "A simple method for generating and sharing pseudo-random functions, with applications to clipper-like escrow systems," in *CRYPTO*, vol. 963, 1995, pp. 185–196.

[36] M. Naor, B. Pinkas, and O. Reingold, "Distributed pseudo-random functions and KDCs," in *EUROCRYPT*, 1999, pp. 327–346.

[37] M. Chase and S. S. M. Chow, "Improving privacy and security in multi-authority attribute-based encryption," in *CCS*, 2009, pp. 121–130.

[38] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl, "Correlated pseudorandom functions from variable-density LPN," in *FOCS*, 2020, pp. 1069–1080.

TABLE 8. RMSE and per-client communication cost (Comm.) in MB for rating prediction tasks (RMSE measures the deviation between the predicted and actual ratings)

		ML1M		ML10M		ML25M		Yelp	
		RMSE	Comm.	RMSE	Comm.	RMSE	Comm.	RMSE	Comm.
MF	Bit8Quant	0.914±0.000	0.25	0.872±0.001	0.69	0.870±0.000	4.06	1.050±0.001	6.07
	TernQuant	0.916±0.002	0.13	0.873±0.001	0.35	0.874±0.000	2.03	1.050±0.001	3.04
	Ours	**0.903**±0.002	0.04	**0.868**±0.003	0.04	**0.864**±0.002	0.04	1.050±0.001	0.03
NCF	Bit8Quant	0.898±0.002	0.13	0.832±0.004	0.53	0.820±0.001	3.06	1.035±0.002	3.83
	TernQuant	0.901±0.007	0.07	0.833±0.001	0.27	0.825±0.002	1.54	1.036±0.004	1.92
	Ours	**0.897**±0.006	**0.02**	**0.819**±0.003	**0.03**	**0.786**±0.008	**0.04**	**1.035**±0.001	**0.02**

TABLE 9. Utility and online per-client communication cost (Comm.) in MB (per FL round) for item ranking tasks: HR@10 refers to hit rate at top-10, and NDCG@10 refers to normalized discounted cumulative gain at top-10. Higher HR@10 and NDCG@10 indicate better ranking quality.

	Beauty			Sport			Game		
	HR@10	NDCG@10	Comm.	HR@10	NDCG@10	Comm.	HR@10	NDCG@10	Comm.
				Caser					
Bit8Quant	0.553±0.002	0.372±0.002	33.44	0.584±0.000	0.410±0.001	63.40	**0.542**±0.001	0.366±0.002	43.70
TernQuant	0.559±0.002	0.376±0.000	16.77	0.583±0.001	0.405±0.001	31.75	0.540±0.003	**0.370**±0.002	21.90
Ours	**0.561**±0.002	0.376±0.001	**0.41**	**0.588**±0.000	**0.411**±0.002	**0.41**	0.540±0.001	0.369±0.000	**0.42**
				SASRec					
Bit8Quant	**0.562**±0.004	0.372±0.006	11.20	0.590±0.002	0.413±0.001	21.12	0.533±0.007	0.356±0.010	14.60
TernQuant	0.557±0.003	0.379±0.004	5.68	0.588±0.002	0.415±0.001	10.64	0.530±0.004	0.360±0.002	7.38
Ours	0.561±0.002	**0.379**±0.001	**0.27**	**0.592**±0.003	**0.415**±0.001	**0.27**	**0.542**±0.003	**0.371**±0.004	**0.27**

[39] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[40] A. Krizhevsky, "Learning multiple layers of features from tiny images," Univ. Toronto, Technical Report, 2009.

[41] Y. Le and X. Yang, "Tiny ImageNet visual recognition challenge," Stanford University, Tech. Rep., 2015, CS231n Course Project Report. [Online]. Available: http://cs231n.stanford.edu/reports/2015/pdfs/yle_project.pdf

[42] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach," arXiv 1907.11692, 2019.

[43] A. Warstadt, A. Singh, and S. R. Bowman, "Neural network acceptability judgments," *Trans. Assoc. Comput. Linguistics*, vol. 7, pp. 625–641, 2019.

[44] W. B. Dolan and C. Brockett, "Automatically constructing a corpus of sentential paraphrases," in *International Workshop on Paraphrasing (IWP@IJCNLP)*, 2005.

[45] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," in *ICLR*, 2022.

[46] F. M. Harper and J. A. Konstan, "The MovieLens datasets: History and Context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, pp. 19:1–19:19, 2016.

[47] Yelp, "Yelp dataset," https://www.yelp.com/dataset, 2015.

[48] Y. Koren, R. M. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[49] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. Chua, "Neural collaborative filtering," in *WWW*, 2017, pp. 173–182.

[50] J. Tang and K. Wang, "Personalized top-n sequential recommendation via convolutional sequence embedding," in *WSDM*, 2018, pp. 565–573.

[51] W. Kang and J. J. McAuley, "Self-attentive sequential recommendation," in *ICDM*, 2018, pp. 197–206.

[52] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors," *J. Math. Cryptol.*, vol. 9, no. 3, pp. 169–203, 2015.

[53] H. Asi, V. Feldman, H. Keller, G. N. Rothblum, and K. Talwar, "PREAMBLE: Private and efficient aggregation of block sparse vectors and applications," IACR Cryptol. ePrint Arch. 2025/490, 2025.

[54] A. Agarwal, E. Boyle, N. Gilboa, Y. Ishai, M. Kelkar, and Y. Ma, "Compressing unit-vector correlations via sparse pseudorandom generators," in *CRYPTO Part VIII*, 2024, pp. 346–383.

[55] L. K. L. Ng and S. S. M. Chow, "GForce: GPU-friendly oblivious and rapid neural network inference," in *USENIX Security Symposium*, 2021, pp. 2147–2164.

[56] A. Y. L. Kei and S. S. M. Chow, "SHAFT: Secure, handy, accurate, and fast transformer inference," in *NDSS*, 2025.

[57] O. Goldreich, "Candidate one-way functions based on expander graphs," in *Studies in Complexity and Cryptography*, 2011, vol. 6650, pp. 76–87.

[58] B. Applebaum, "Cryptographic hardness of random local functions - survey," *Comput. Complex.*, vol. 25, no. 3, pp. 667–722, 2016.

[59] E. Boyle, N. Gilboa, and Y. Ishai, "Breaking the circuit size barrier for secure computation under DDH," in *CRYPTO Part I*, 2016, pp. 509–539.

[60] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *ICLR (Poster)*, 2019.

[61] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR (Poster)*, 2015.

Appendix A. Further Cryptography Background

Definition 5 (Local PRG [57], [58])**.** A d-local $\mathsf{PRG}\colon \Sigma_{\mathsf{in}}^k \to \Sigma_{\mathsf{out}}^n$ with dependency pattern $P\colon [n] \to [k]^d$ should be

- **Pseudorandom**: For a random seed $sd \leftarrow \Sigma_{\mathsf{in}}^k$, its output $y = \mathsf{PRG}(sd) \in \Sigma_{\mathsf{out}}^n$ is pseudorandom.
- **d-local**: Let $sd = (sd_1, \ldots, sd_k)$, $y = (y_1, \ldots, y_n) \leftarrow \mathsf{LPRG}(sd)$. For $j \in [n]$, y_j only depends on $(sd_{i_1}, \ldots, sd_{i_d})$, where $P(j) = (i_1, \ldots, i_d)$.

For notational convenience, for $j \in [n]$, we use $\mathsf{PRG.Eval}(sd, j)$ to denote y_j, *i.e.*, its j-th output.

Definition 6 (Low-degree Sparse PRG [54])**.** Let $\mathcal{R}$ be a ring. A degree-d $\mathsf{SPRG}\colon \mathcal{R}^k \to \mathcal{R}^{nB}$ is a tuple $(\mathsf{KeyGen}, \mathsf{Eval}, P)$.

- $\mathsf{KeyGen}(1^\lambda)$: (a PPT algorithm) Output a seed $\mathsf{ssd} \in \mathcal{R}^k$.
- $\mathsf{Expand}(\mathsf{ssd})$: (an efficient deterministic algorithm) Output $(y_1, \ldots, y_n)$, where $y_i \in \mathcal{R}^B$.

For notational convenience, for $j_1 \in [B], j_2 \in [n]$, we use $\mathsf{Eval}(\mathsf{ssd}, j_1, j_2)$ to denote $y_{j_1,j_2} \in \mathcal{R}$, *i.e.*, its (j_1, j_2)-th output. A degree-d sparse PRG requires that

- **Pseudorandom**: For $\mathsf{ssd} \leftarrow \mathsf{KeyGen}(1^\lambda)$, $y_j = \mathsf{Eval}(\mathsf{ssd}, j) \in \mathcal{R}^B$ is indistinguishable from a random unit vector, *i.e.*, $(0, \ldots, 1, \ldots, 0) \in \mathcal{R}^B$ with $1 \in \mathcal{R}$ at a random position.
- **degree-d**: For any $j_1 \in [B], j_2 \in [n]$, $f(\mathsf{ssd}) := \mathsf{Eval}(\mathsf{ssd}, j_1, j_2)$ can be computed as a degree-d multivariate polynomial on input $\mathsf{ssd} \in \mathcal{R}^k$.

A d-local PRG can be transformed into a degree-d SPRG over any ring $\mathcal{R}$.

Lemma 1 (Transformation from d-local PRG to degree-d sparse PRG [54])**.** *Let $G\colon \Sigma_{\mathsf{in}}^k \to \Sigma_{\mathsf{out}}^n$ be a d-local PRG. Let $\mathcal{R}$ be any ring. Then there exists a degree-d sparse PRG $G'\colon \mathcal{R}^{k|\Sigma_{\mathsf{in}}|} \to \mathcal{R}^{n|\Sigma_{\mathsf{out}}|}$. Moreover, there exists a seed conversion algorithm*

- $\mathsf{ConvertSeed}(sd)$*: (an efficient deterministic algorithm) On input the seed $sd \in \Sigma_{\mathsf{in}}^k$ for d-local PRG, output the seed for degree-d sparse PRG,* $\mathsf{ssd} \in \mathcal{R}^{k|\Sigma_{\mathsf{in}}|}$.

For $\mathsf{ssd} \leftarrow \mathsf{ConvertSeed}(sd)$*, any $j \in [n]$, we have*

$$\vec{\boldsymbol{y}} = \vec{\boldsymbol{e}}_y$$

where $\vec{\boldsymbol{y}} \leftarrow G'.\mathsf{Eval}(\mathsf{ssd}, j) \in \mathcal{R}^{|\Sigma_{\mathsf{out}}|}$, and $y \leftarrow G.\mathsf{Eval}(sd, j) \in \Sigma_{\mathsf{out}}$, $\vec{\boldsymbol{e}}_y \in \mathcal{R}^{|\Sigma_{\mathsf{out}}|}$ is the basis vector, i.e., equal to $1 \in \mathcal{R}$ at index y (i.e., the order of y in Σ_{out}) and 0 elsewhere.

Definition 7 (Homomorphic Secret Sharing [59])**.** HSS is a form of secret sharing that enables homomorphic evaluation on shares, resulting in additive shares of the computation output. An m-party HSS scheme for degree-d computation is a pair $(\mathsf{Enc}, \mathsf{DistEval})$ defined as follows.

- $\mathsf{Enc}(1^\lambda, x)$: (a PPT algorithm) On input security parameter 1^λ, input x, output $(x_j)_{j\in[m]}$.
- $\mathsf{DistEval}(x_j, f(\cdot))$: (an efficient deterministic algorithm) On input x_i, and a polynomial f of degree d, output the additive share of $f(x)$.

HSS requires

- **Correctness**: For $(x_j)_{j\in[m]} \leftarrow \mathsf{Enc}(1^\lambda, x)$, and a polynomial f of degree d, it holds that
$$\sum_{j\in[m]} \mathsf{DistEval}(x_j, f(\cdot)) = f(x)$$
- **Security**: Any subset of the shares $(x_j)_{j\in[m]}$ leaks nothing about x.

We use HSS that supports the evaluation of low-degree (*e.g.*, ≤ 5) multivariate polynomials f on secretly shared input, which can be concretely efficient in practice [54].

Appendix B. Low-Degree Random Sparse PRGs

We now give the definition of low-degree random sparse SPG (RSPRG), which is an intermediate primitive for our construction of DynF (Figure 3).

Definition 8 (Low-degree Random Sparse PRG (RSPRG))**.** Let $\mathcal{R}$ be a ring. A degree-d RSPRG from Σ_{in}^k to $[N]^n$ is a tuple $(\mathsf{KeyGen}, \mathsf{CoreEval}, \mathsf{SparseEval})$.

- $\mathsf{KeyGen}(1^\lambda, r \in \mathcal{R})$: (a PPT algorithm) Output a main seed $sd \in \Sigma_{\mathsf{in}}^k$, and a sparse seed $\mathsf{ssd} \in \mathcal{R}^k$.
- $\mathsf{CoreExpand}(sd)$: (an efficient deterministic algorithm) On input the main seed sd, output $y \in [N]^n$.
- $\mathsf{SparseEval}(\mathsf{ssd}, x \in [n])$: (an efficient deterministic algorithm) On input the sparse seed ssd, output $(\vec{\boldsymbol{p}}, \vec{\boldsymbol{r}}) \in \mathcal{R}^N \times \mathcal{R}^N$.

For notational convenience, for $x \in [n]$, we use $\mathsf{CoreEval}(sd, x)$ to denote the x-th output of $\mathsf{CoreExpand}(sd)$. A degree-d RSPRG should be

- **Pseudorandom**: Let $(sd, *) \leftarrow \mathsf{KeyGen}(1^\lambda, r)$, $r \in \mathcal{R}$, $\mathsf{CoreExpand}(sd)$ is pseudorandom over $[N]^n$.
- **Correct**: For any $r \in \mathcal{R}$, let $(sd, \mathsf{ssd}) \leftarrow \mathsf{KeyGen}(1^\lambda, r)$, for any $x \in [n]$, $y \leftarrow \mathsf{CoreEval}(sd, x) \in [N]$, $(\vec{\boldsymbol{p}}, \vec{\boldsymbol{r}}) \leftarrow \mathsf{SparseEval}(\mathsf{ssd}, x) \in \mathcal{R}^N \times \mathcal{R}^N$, it holds that
$$\vec{\boldsymbol{p}} = \vec{\boldsymbol{e}}_y, \text{ and } \vec{\boldsymbol{r}} = r \cdot \vec{\boldsymbol{e}}_y$$
where $\vec{\boldsymbol{e}}_y \in \mathcal{R}^N$ is the basis vector, *i.e.*, equal to $1 \in \mathcal{R}$ at y, 0 otherwise.
- **degree-d**: For any $x \in [n]$, $f(\mathsf{ssd}) := \mathsf{SparseEval}(\mathsf{ssd}, x)$ can be computed as a degree-d multivariate polynomial on input $\mathsf{ssd} \in \mathcal{R}^k$.

The construction of RSPRG is formally described in Figure 13. One might view similar construction ideas as implicit [54] in the context of authenticated unit-vector pairs.

Theorem 5. *The construction in Figure 13 is a degree-d random sparse PRG (Definition 8) from Σ_{in}^d to $[N]^d$.*

Proof. The correctness and low-degree computation can be easily verified. Pseudorandomness follows from that of the underlying local PRG. □

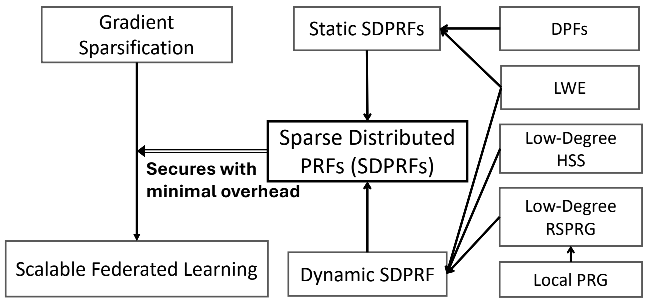

Figure 12. Roadmap

Appendix C. Security Proofs

C.1. Proof of Theorem 1

Proof. **Correctness**: Let $(mk, (dk_j)_{j\in[m]}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. For any $x \in \mathcal{X}$, let $(p, y) \leftarrow \mathsf{CoreEval}(mk, x)$, $(\vec{\boldsymbol{p}}_j, \vec{\boldsymbol{y}}_j) \leftarrow \mathsf{DistEval}(dk_j = (k_j, k'_j), x)$, then it holds that

$$\sum_{j\in[m]} \vec{\boldsymbol{p}}_j = \sum_{j\in[m]} \mathsf{DPF.EvalAll}(k_j) = \vec{\boldsymbol{e}}_p$$

$$\begin{aligned}\sum_{j\in[m]} \vec{\boldsymbol{y}}_j &= \sum_{j\in[m]} \left\langle H_1(x), \mathsf{DPF.EvalAll}(k'_j)\right\rangle \\ &= \langle H_1(x), mk \cdot \vec{\boldsymbol{e}}_p\rangle \\ &= (0, \ldots, y + \chi, \ldots, 0) \text{ (nonzero position at } p)\end{aligned}$$

where $\vec{\boldsymbol{e}}_p \in \mathcal{R}^n$ is the basis vector, *i.e.*, equal to $1 \in \mathcal{R}$ at index p and 0 elsewhere, and χ is LWE noise. □

Security: We proceed by a sequence of hybrids.

Hybrid G_1: This is the real world, where adversary $\mathcal{A}$ gets $\{dk_j\}_{j\in\mathcal{S}}$, and oracle access to $F_{mk}(x) := \mathsf{CoreEval}(mk, x)$, where $\mathcal{S}$ is a strict subset of $[m]$.

Hybrid G_2: This is the intermediate world, where adversary $\mathcal{A}$ gets $\{r_j\}_{j\in\mathcal{S}}$, and oracle access to $F_{mk}(x) := \mathsf{CoreEval}(mk, x)$, where $\mathcal{S}$ is a strict subset of $[m]$.

Hybrid G_3: This is the ideal world, where the adversary $\mathcal{A}$ gets $\{r_j\}_{j\in\mathcal{S}}$, and the oracle access to a random function $R(\cdot)$, where r is a random value, $\mathcal{S}$ is a strict subset of $[m]$.

G_1 and G_2 are indistinguishable due to the security of DPF. G_2 and G_3 are indistinguishable because $(H_1(x_i), \langle H_1(x_i), \boldsymbol{s}\rangle + \mathsf{Gau}(H_2(x_i\|mk), B))$ is indistinguishable from $(H_1(x_i), r_i)$ by the LWE assumption.

C.2. Proof of Theorem 2

Proof. The correctness can be verified similarly to Appendix C.1. The security follows from the HSS that one can learn nothing from any subset of distributed keys with the pseudorandomness of the RSPRG and that of the CoreEval as a pseudorandom function. □

Appendix D. Client Registration Protocol

The client registration process is presented in Figure 14, where the client distributes the keys of distributed sparse PRFs to the servers and retains the master key locally.

Appendix E. Experiment

E.1. Other Experimental Setting

The experiments are conducted on a 96-core Ubuntu Linux 20.04 server with 128GB RAM and L40 driver.

Each utility score reported in the tables is an average of 4 runs of the experiment. The training hyperparameters for each task are provided as follows.

Image classification: All models are updated using stochastic gradient descent (SGD) with a learning rate of 0.01. We train the models for 200 FL rounds on CIFAR-10, and 500 FL rounds on CIFAR-100 and Tiny ImageNet. Datasets are partitioned non-IID across clients, each holding 64 samples.

Language understanding: We divide the datasets randomly across clients, each holding 30 samples. We configure LoRA [45] with rank $r = 8$ and scaling factor $\alpha = 32$. Each model is trained for 500 FL rounds with a learning rate of 0.00001 using AdamW optimizer [60].

Recommendation systems: For MF, we use an embedding size of 64 and a learning rate of 0.01 across all datasets. For NCF, embedding sizes are set to 16, 24, 24, and 20 for ML1M, ML10M, ML25M, and Yelp, respectively, with a learning rate of 0.001. We train the models for 300, 500, 1000, and 300 FL rounds, respectively, for ML1M, ML10M, ML25M, and Yelp. For Caser and SASRec, we adopt the hyperparameters from prior work [50], [51]. The two sequence models are trained for 500 FL rounds on the three Amazon datasets. Each model is updated using the Adam optimizer [61].

E.2. Utility Analysis on Language Understanding

We evaluate the ACC and AUC on language understanding task under varying sparsification ratio ρ in Figure 11. The accuracy loss remains minimal ($\leq$1%) under $\rho \geq 0.3$ for both datasets. Under sparsification ratio $\rho = 0.1$, the performance degradation rises to around 7.3% for MRPC and 1.4% for CoLA. To retain utility while optimizing communication efficiency, we choose sparsification ratio $\rho = 0.5$ in Section 6.4.

E.3. Utility Analysis on Image Classification (IID)

We evaluate the utility of the image classification task under the IID setting in Table 10. With a sparsification ratio of $\rho = 0.2$, the utility loss averages only 0.9% compared to full gradient transmission. For smaller ρ, the loss increases from 1.6% at $\rho = 0.15$ to 6.5% at $\rho = 0.05$, while achieving greater communication savings.

TABLE 10. ACCURACY ON IMAGE CLASSIFICATION UNDER VARYING TOP-k SPARSIFICATION RATIO ρ IN IID SETTING

ρ	1	0.20	0.15	0.10	0.05
CIFAR-10	0.912	0.911	0.909	0.908	0.902
CIFAR-100	0.654	0.641	0.639	0.624	0.596
Tiny ImageNet	0.542	0.538	0.529	0.507	0.489

Construction of degree-d RSPRG

Public parameters:

- Dimension N, degree $d \leq 5$, a ring $\mathcal{R}$.

Building blocks:

- d-local PRG: $\Sigma_{\mathsf{in}}^{k} \to [N]^{k^d}$ (Definition 5).

Construction:

- $\mathsf{KeyGen}(1^\lambda, r \in \mathcal{R})$:
  - Sample a random seed $\mathsf{sd} \leftarrow \Sigma_{\mathsf{in}}^{k}$.
  - $\mathsf{ssd}' \leftarrow \mathsf{ConvertSeed}(\mathsf{sd}) \in \mathcal{R}^{k'}$, where $k' = k|\Sigma_{\mathsf{in}}|$ (Lemma 1).
  - $\mathsf{rssd}' \leftarrow r \cdot \mathsf{ssd}' \in \mathcal{R}^{k'}$ (scalar-vector product)
  - $\mathsf{ssd} \leftarrow (\mathsf{ssd}', \mathsf{rssd}') \in \mathcal{R}^{2k'}$.
  - Output sd, ssd.
- $\mathsf{CoreEval}(sd, x \in [k^d])$:
  - $y \leftarrow \mathsf{PRG.Eval}(\mathsf{sd}, x) \in [N]$.
  - Output $y \in [N]$.
- $\mathsf{SparseEval}(\mathsf{ssd} \in \mathcal{R}^{2k}, x \in [k^d])$:
  - Parse $\mathsf{ssd} = (\mathsf{ssd}', \mathsf{rssd}')$, where $\mathsf{ssd}', \mathsf{rssd}' \in \mathcal{R}^{k'}$.
  - For $i \in [N]$:
    * Let $G' \colon \mathcal{R}^{k'} \to \mathcal{R}^{Nk^d}$ be the sparse PRG induced by the d-local PRG (Lemma 1).
    * Let $f_i \colon \mathcal{R}^{k'} \to \mathcal{R}$ be the degree-d polynomial for computing the i-th output of G', with input-output dependency map $P(i)$. The i-th output of G' can be represented as

$$f(\boldsymbol{a})_i = \sum_{(p_1,\ldots,p_d)\in P(i)} \left(\prod_{t=1}^{d} \boldsymbol{a}_{p_t} \right).$$

    * Compute $p_i \leftarrow f_{x,i}(\mathsf{ssd}') \in \mathcal{R}$.
    * Compute $y_i \in \mathcal{R}$:

$$y_i \leftarrow \sum_{\in P(x,i)} \left(\mathsf{rssd}'_{p_1} \cdot \prod_{t=2}^{d} \mathsf{ssd}'_{p_t} \right).$$

  - $\vec{\boldsymbol{p}} \leftarrow (p_1, \ldots, y_n) \in \mathcal{R}^N$.
  - $\vec{\boldsymbol{y}} \leftarrow (y_1, \ldots, y_n) \in \mathcal{R}^N$.
  - Output $(\vec{\boldsymbol{p}}, \vec{\boldsymbol{y}}) \in \mathcal{R}^N \times \mathcal{R}^N$.

Figure 13. Our degree-d random sparse PRG (Definition 8)

Client registration subroutine: Register

Global parameter:

- Model size N; Number of servers n_s.

Client public parameter:

- Bandwidth upper bound B (of prospective training round).
- Selected servers $\mathcal{S} \subseteq [n_s]$, where $|\mathcal{S}| = n'_s$.
- Counter cnt, count bound M.

Client private parameter:

- Static sparsity pattern $\mathcal{SI} \subset [N]$.

Building blocks:

- N-dimensional StaF, with size-s_1 master and size-s_2 distributed keys.
- N-dimensional DynF for an input domain of size M, with size-s_3 master key, and s_4 distributed key size.

On invocation:

1) Initialize counter cnt $\leftarrow 0$.
2) **Static sparsity key generation**
   - Initialize empty lists $\mathsf{StaDecKey}_1, \ldots, \mathsf{StaDecKey}_{n'_s}$.
   - For each $i \in \mathcal{SI}$:
     - $mk_i, \{dk_{i,j}\}_{j\in\mathcal{S}} \leftarrow \mathsf{StaF.KeyGen}(1^\lambda, i)$.
     - $\mathsf{StaDecKey}_j.\mathsf{append}(dk_{i,j})$, for $j \in \mathcal{S}$.
   - Store $\mathsf{StaEncKey} = (mk_1, \ldots, mk_{|\mathcal{SI}|})$.
3) **Dynamic sparsity key generation**
   - Set M such that $s_4(M, N) \cdot B \cdot n'_s \leq M$.
   - $mk, \{dk_j\}_{j\in\mathcal{S}} \leftarrow \mathsf{DynF.KeyGen}(1^\lambda)$.
   - $mk', \{dk'_j\}_{j\in\mathcal{S}} \leftarrow \mathsf{DynF.KeyGen}(1^\lambda)$.
   - $\mathsf{DynEncKey} \leftarrow mk$.
   - $\mathsf{DynDecKey}_j \leftarrow dk_j$ for $j \in \mathcal{S}$.
   - $\mathsf{DynKey}' = mk', \{dk'_j\}_{j\in\mathcal{S}}$
   - Store $\mathsf{DynEncKey}$, DynKey'.
4) **Submit registration**
   - For selected server $j \in \mathcal{S}$:
     - $\mathsf{PDecKey}_j \leftarrow (\mathsf{StaDecKey}_j, \mathsf{DynDecKey}_j)$.
     - Send $\mathsf{PDecKey}_j$ to server j.

Client cost:

- **Registration communication:**
  - For static sparsity: $n'_s \cdot |\mathcal{SI}| \cdot (s_1 + s_2)$.
  - For dynamic sparsity: $n'_s \cdot s_4$.
- **Client storage:**
  - For static sparsity: s_0.
  - For dynamic sparsity: $2s_3 + n'_s \cdot s_4$.

Figure 14. Client registration process

Appendix F. Meta-Review

The following meta-review was prepared by the program committee for the 2026 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

F.1. Summary

The paper proposes a secure aggregation scheme for federated learning that is compatible with gradient sparsification, thereby increasing communication efficiency. The proposal involves multiple (≥ 2) benign, non-colluding servers that each aggregate one share of the clients' gradients. The paper proposes a so-called "sparse distributed pseudorandom function" — a new cryptographic primitive — to allow each client to securely share its top-k gradients with the aggregation servers in a way that is indistinguishable from randomness.

F.2. Scientific Contributions

- Provides a Valuable Step Forward in an Established Field

F.3. Reasons for Acceptance

1) Compared to related proposals, the proposal reduces communication overhead by a considerable margin.
2) The empirical evaluation is thorough and encompasses different ML applications.
3) Much of the paper is well written and easy to follow.

F.4. Noteworthy Concerns

1) In many of the presented results, it is not clear if the results are statistically significant.

F.5. Response to the Meta-Review

We believe the noteworthy concern originates from the comment that *"In many of the presented results, the numbers are too similar to state that one is better than the other. Although the average might be higher, high standard deviations undermine the consistency of the results."*

The shepherd suggested: *"I'd encourage you to include evidence to demonstrate that your results are statistically significant, for example, through regression analysis."*

All reported standard deviations pertain to utility or accuracy metrics. Our goal is to cut communication while remaining as accurate as the plaintext baseline within a small margin. We do not claim higher utility or accuracy. Accordingly, we test non-inferiority to the baseline using paired repeated runs and report confidence intervals for paired differences. These tests determine whether our utility remains within the specified margin while achieving the reported communication savings. We think a regression analysis is not a must for supporting this claim.