# Thinking Forward: Memory-Efficient Federated Finetuning of Language Models

**Kunjal Panchal**
University of Massachusetts
Amherst, MA 01003-9264
`kpanchal@umass.edu`

**Nisarg Parikh**
University of Massachusetts
Amherst, MA 01003-9264
`nkparikh@umass.edu`

**Sunav Choudhary**
Adobe Research
Bangalore, India 560103
`schoudha@adobe.com`

**Lijun Zhang**
University of Massachusetts
Amherst, MA 01003-9264
`lijunzhang@cs.umass.edu`

**Yuriy Brun**
University of Massachusetts
Amherst, MA 01003-9264
`brun@cs.umass.edu`

**Hui Guan**
University of Massachusetts
Amherst, MA 01003-9264
`huiguan@cs.umass.edu`

## Abstract

Finetuning large language models (LLMs) in federated learning (FL) settings has become increasingly important as it allows resource-constrained devices to finetune a model using private data. However, finetuning LLMs using backpropagation requires excessive memory (especially from intermediate activations) for resource-constrained devices. While Forward-mode Auto-Differentiation (AD) can significantly reduce memory footprint from activations, we observe that directly applying it to LLM finetuning results in slow convergence and poor accuracy. In this paper, we introduce SPRY, an FL algorithm that splits trainable weights of an LLM among participating clients, such that each client computes gradients using Forward-mode AD that are closer estimations of the true gradients. SPRY achieves a low memory footprint, high accuracy, and fast convergence. We formally prove that the global gradients in SPRY are unbiased estimators of true global gradients for homogeneous data distributions across clients, while heterogeneity increases bias of the estimates. We also derive SPRY's convergence rate, showing that the gradients decrease inversely proportional to the number of FL rounds, indicating the convergence up to the limits of heterogeneity. Empirically, SPRY reduces the memory footprint during training by 1.4–7.1$\times$ in contrast to backpropagation, while reaching comparable accuracy, across a wide range of language tasks, models, and FL settings. SPRY reduces the convergence time by 1.2–20.3$\times$ and achieves 5.2–13.5% higher accuracy against state-of-the-art zero-order methods. When finetuning Llama2-7B with LoRA, compared to the peak memory consumption of 33.9GB of backpropagation, SPRY only consumes 6.2GB of peak memory. For OPT13B, the reduction is from 76.5GB to 10.8GB. SPRY makes feasible previously impossible FL deployments on commodity mobile and edge devices. Our source code is available for replication at `https://github.com/Astuary/Spry`.

## 1 Introduction

In cross-device federated learning (FL), thousands of edge devices (called *clients*) collaborate through an orchestrator (called *server*) to jointly train a machine learning (ML) model [1, 2]. In each round of FL, the server sends an ML model to participating clients, who then update the model weights for several epochs on their individual data and send the new weights back to the server. The server aggregates the weights to update the model and initiates the next round of FL training. Due to the

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

inherent privacy-preserving nature of FL, it has been adopted in many privacy-sensitive domains, such as healthcare [3], IoT [4, 5], and e-commerce [6].

In parallel, large language models (LLMs) have demonstrated impressive performance on natural language processing tasks [7, 8], creating a surge of interest in finetuning LLMs in FL settings [9, 10]. However, the problem is challenging in practice because of the memory requirements in finetuning LLMs. LLMs can have billions of weights, and finetuning them with backpropagation requires dozens of GBs of memory. These requirements easily overwhelm edge devices, particularly mobile phones with limited memory. The memory footprint of finetuning LLMs mainly comes from model weights and their gradients, optimizer states, and intermediate activations.

There are three categories of existing algorithms that reduce the memory footprint of finetuning LLMs: parameter-efficient finetuning (PEFT) [11–16], quantization [17], and zero-order gradient estimation methods [18–20]. Although PEFT and quantization can reduce the memory consumption from parameters and optimizer states, the memory consumed by intermediate activations remains a significant bottleneck because these methods still use backpropagation to finetune LLMs. Backpropagation requires storing all intermediate activations during the forward pass to estimate gradients in the backward pass. For example, finetuning a 4-bit quantized Llama2-7B model [21] with LoRA techniques [12] requires $\sim$33.9GB of RAM, with 83.8% used for intermediate activations. Zero-order methods leverage finite difference [22] to estimate gradients and thus reduce the memory consumption from intermediate activations [19, 20, 23]. However, these methods suffer from slow convergence and poor model quality because the accumulation of truncation and round-off errors [24], a fundamental issue of finite difference, leads to noisy estimation of weight gradients.

*Forward-mode Auto-Differentiation (AD)* [24, 23] has the potential to address the memory consumption problems of backpropagation without introducing the round-off errors of finite difference. It estimates gradients by computing a Jacobian-vector product (`jvp`) based on random perturbations of weights during the forward pass, alleviating the need to store all intermediate activations, similar to zero-order methods. `jvp` represents how much changing the weights in the direction of a random perturbation affects the outputs. However, simply replacing backpropagation with Forward-mode AD in FL settings does not produce convergence speed and accuracy performance comparable to established federated optimizers based on backpropagation, such as FEDAVG [1], FEDYOGI [25], FEDSGD [1]. Forward gradients are *computationally inefficient* and *inaccurate* in estimating true gradients when the number of trainable weights is large. We empirically observed that, for LLMs finetuned with the LoRA technique, Forward-mode AD suffers from 8.3–57.2% accuracy loss and is 1.4–3.9$\times$ slower to converge for models whose number of trainable weights exceed approximately 1.15M (see Appendix G).

In this paper, we propose SPRY[1], an FL algorithm for finetuning LLMs using Forward-mode AD while achieving low memory footprint, high accuracy, and fast convergence. SPRY tackles the shortcomings of Forward-mode AD by splitting the trainable weights among participating clients per FL round. SPRY improves computation efficiency as each client only needs to perturb a small fraction of trainable weights to derive their gradients, reducing the number of computations in each forward pass. SPRY achieves higher accuracy and faster convergence because the smaller number of trainable weights for each participating client allows computing gradients that are closer estimations of the true gradients. In contrast to zero-order methods where one training iteration requires 20–100 forward passes, each with a different perturbation [20, 19] to estimate weight gradients well, SPRY allows computing weight gradients from only one forward pass per iteration for each client. Unlike split learning [26], SPRY does not need to transfer intermediate activations among clients. The union of the partial weights trained from each participating client in an FL round updates all the trainable weights of the language model. Since only a subset of weights is finetuned per client, SPRY also saves client-to-server communication bandwidth.

We formally prove that the global gradients aggregated on the server side in SPRY are unbiased estimators of the true global gradients in case of homogeneous data distributions across clients, while the heterogeneity increases the bias of the estimations. We also derive the convergence rate of SPRY, showing that the norm of global gradients decreases linearly with the inverse of the number of FL rounds. We further discuss how configurations in SPRY affect the convergence behavior of the algorithm and empirically validate the theoretical analysis.

[1]Named after its light memory consumption and speed.

We empirically evaluate SPRY's memory efficiency, accuracy, computation efficiency, and communication efficiency through experiments on a wide range of language tasks, models, and FL settings. SPRY achieves within 0.6–6.2% of the accuracy of the best-performing FL backpropagation, with 1.4–7.1× less memory consumption for each client and comparable time to convergence. SPRY also outperforms zero-order-based baselines with 5.2–13.5% higher accuracy, an average of 1.5–28.6× faster per-round computation time, and 1.2–20.3× faster convergence. We also compare SPRY's communication efficiency to that of FEDAVG (per-epoch communication) and FEDSGD (per-iteration communication). For communication frequency of per-epoch, SPRY reduces the number of model weights sent from a client to the server by $M$ times, where $M$ is the number of participating clients per round. For per-iteration communication frequency, each client of SPRY only needs to send back a scalar to the server, fixing the client-to-server total communication cost to $M$ scalar values.

We make the following contributions:

1. SPRY, the first work that demonstrate the potential of Forward-mode AD for finetuning language models (with 18M to 13B parameters) in FL settings with low memory footprint, high accuracy, and fast convergence.
2. A federated optimization strategy that only requires a single forward pass per batch on each client to finetune a language model.
3. A theoretical analysis of how SPRY's global gradients estimate true gradients based on the heterogeneity of FL clients, and a proof that SPRY's convergence is linearly dependent on the number of FL rounds when a client's learning rate is inversely proportional to the size of perturbations and client data heterogeneity.
4. An empirical evaluation shows that SPRY consumes 1.4–7.1× less memory than its backpropagation-based counterparts, and converges 1.2–20.3× faster with 5.2–13.5% higher accuracy compared to its zero-order counterparts.

## 2 Forward-mode Automatic Differentiation

This section presents the background on Forward-mode Auto-Differentiation (AD) necessary to follow the work. Related works on zero-order optimization methods, Forward-mode AD, and FL for LLMs are discussed in detail in Appendix A.

Forward-mode AD computes gradients by measuring how changes in model weights, in the direction of a random perturbation, affect the loss. Since these gradients are derived from a forward pass, they are referred to as *forward gradients* [23]. In contrast, backpropagation (also Reverse-mode AD) calculates a direction to adjust weights in, to decrease the loss. Formally, for each training iteration, given the trainable weights $\boldsymbol{w}$, Forward-mode AD generates a random perturbation $\boldsymbol{v}$ whose size is the same as $\boldsymbol{w}$. In one forward pass, given training data $\mathcal{D}$, Forward-mode AD computes the value of the objective function $f(\boldsymbol{w};\mathcal{D})$ and the Jacobian-vector product (`jvp`) as follows:

$$\mathbf{J}_f\boldsymbol{v} = \nabla f_{\boldsymbol{v}}(\boldsymbol{w};\mathcal{D}) = \begin{bmatrix}\frac{\partial f(\boldsymbol{w};\mathcal{D})}{\partial w_1} & \dots & \frac{\partial f(\boldsymbol{w};\mathcal{D})}{\partial w_d}\end{bmatrix}\begin{bmatrix}v_1 & \dots & v_d\end{bmatrix}^T. \tag{1}$$

This `jvp` is a scalar for neural networks since the output of the objective function $f$ is a scalar. Multiplying `jvp` with the perturbation $\boldsymbol{v}$ gives us the unbiased estimate of true gradients [23],

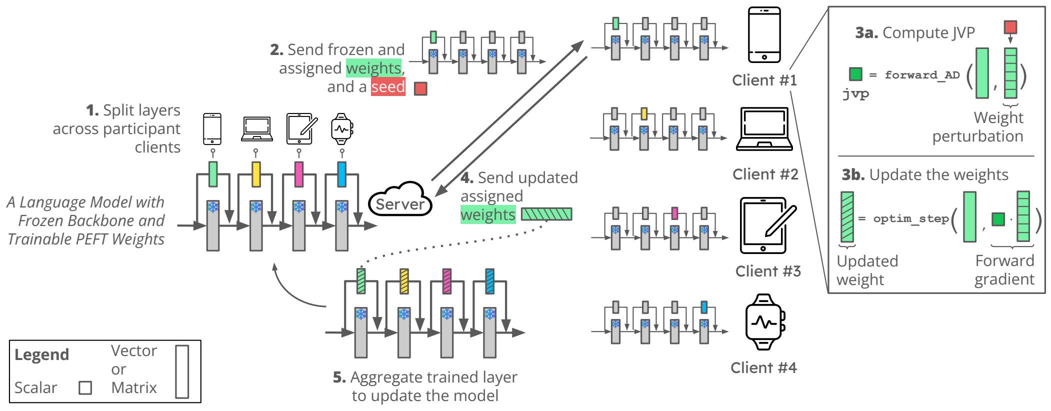
$$\nabla F(\boldsymbol{w}) = \mathbb{E}_{\boldsymbol{v}}\left[\mathbf{J}_f\boldsymbol{v}\cdot\boldsymbol{v}\right] = \mathbb{E}_{\boldsymbol{v},\mathcal{D}}\left[\left(\begin{bmatrix}\frac{\partial f(\boldsymbol{w};\mathcal{D})}{\partial w_1} & \dots & \frac{\partial f(\boldsymbol{w};\mathcal{D})}{\partial w_d}\end{bmatrix}\boldsymbol{v}^T\right)\boldsymbol{v}\right] \tag{2}$$

$$= \mathbb{E}_{\mathcal{D}}\begin{bmatrix}\frac{\partial f(\boldsymbol{w};\mathcal{D})}{\partial w_1} & \dots & \frac{\partial f(\boldsymbol{w};\mathcal{D})}{\partial w_d}\end{bmatrix} = \mathbb{E}_{\mathcal{D}}\left[\nabla f(\boldsymbol{w};\mathcal{D})\right]. \tag{3}$$

The partial derivative $\partial f(\boldsymbol{w};\mathcal{D})/\partial\boldsymbol{w}$ is computed by chain rule on intermediate activations in the forward pass [24]. Unlike backpropagation where all the intermediate activations need to be stored during the forward pass, Forward-mode AD only stores the previous layer's activations in the forward pass for the chain rule derivatives. Hence, the memory overhead of deriving gradients would be the size of the largest activation in the forward pass.

## 3 SPRY: Memory-Efficient Federated Finetuning of Language Models

While Forward-mode AD can decrease memory usage during model finetuning, merely substituting backpropagation with Forward-mode AD in FL scenarios often results in poor accuracy and



1. Split layers across participant clients
A Language Model with Frozen Backbone and Trainable PEFT Weights
2. Send frozen and assigned weights, and a seed
Server
4. Send updated assigned weights
5. Aggregate trained layer to update the model
Legend
Scalar
Vector or Matrix
Client #1
Client #2
Client #3
Client #4
3a. Compute JVP
jvp = forward_AD( , )
Weight perturbation
3b. Update the weights
= optim_step( , · )
Updated weight
Forward gradient

Figure 1: Overview of SPRY, a federated learning framework to finetune language models with low memory footprint. The term "PEFT" stands for parameter-efficient fine-tuning.

computational inefficiency. To address this challenge, SPRY recognizes that Forward-mode AD operates more efficiently and yields better gradient estimations when the trainable weight count is minimized. Therefore, SPRY optimizes performance by distributing trainable weights among participating clients, assigning each client a responsibility to compute gradients for only a subset of trainable weights. SPRY is compatible with PEFT methods such as IA3 [15], ADAPTER-based methods [27], BITFIT [28], and LORA [12], which mitigate the memory consumption from gradients and optimizer states. In this work, we focus on LORA due to its demonstrated superiority, as highlighted in Appendix G.

Figure 1 gives an overview of SPRY. It includes 5 main steps: **(1)** At the beginning of each FL round, the server assigns a few trainable layers to each of the participating clients of that round. **(2)** Each client is sent (i) the trainable weights of the layers assigned to it, (ii) frozen weights of the remaining layers if not previously received, and (iii) a scalar seed value. **(3)** On the client side, weight perturbations for the allocated layers are generated based on the received seed. These perturbations are utilized to update the assigned weights through computation of the forward gradients. **(4)** Clients only transmit the updated weights back to the server. **(5)** The server aggregates all the trained layer weights and updates the language model for the subsequent round.

Next, we discuss the two key steps of SPRY in detail: Step (1), where the server assigns trainable weights to the participating clients and Step (3), where each client finetunes the assigned trainable weights using Forward-mode AD.

### 3.1 Assigning Trainable Layers to Clients at the Server-side

To enable closer gradient estimations through forward gradients, the server reduces the number of trainable weights per client by selecting a layer and assigning it to a client in a cyclic manner. With LORA, the server selects a LoRA layer, which consists of a pair of weights ($w_A$ and $w_B$ matrices) for each client. When # trainable layers $>$ # participating clients, each client will be assigned more than one layer. Otherwise, each layer will be assigned to more than one client. The server aggregates the trained weights from each client and updates the model using adaptive optimizers such as FEDYOGI. Adaptive optimizers are shown to be less prone to noisy updates compared to FEDAVG in the literature [25, 29]. The server keeps a mapping of layer names to client IDs, hence it can gather updated layer weights from all clients and update the model.

FL often faces the data heterogeneity issue, where the data distribution of one client differs from another, leading to poor model accuracy. While the primary aim of SPRY does not directly tackle this issue, it seamlessly integrates with existing finetuning-based personalization techniques [30] to mitigate it. SPRY distributes trainable classifier layers to all participating clients, enabling each client to finetune these layers to personalize the jointly trained model.

### 3.2 Finetuning Weights with Forward Gradients on the Client-side

Clients update the assigned trainable weights with gradients estimated through Forward-mode AD. Specifically, each participating client will get a copy of the trainable weights assigned to it and a scalar seed value from the server. Using the seed value, the client generates a random perturbation for each trainable weight, following a normal distribution with a mean of 0 and a standard deviation of 1. Forward gradients are obtained during forward pass, as shown in Eq. 3. The subsequent steps depend on the communication frequency.

**Per-Epoch Communication.** Per-epoch communication means that each client transmits the updated trainable weights to the server after every one or more epochs. Locally, the trainable weights are updated using optimizers such as SGD and ADAM. Only the updated trainable weights are sent back to the server, which reduces communication costs. Each client transmits the weights of $\max\left\{\frac{\text{\#Trainable Layers}}{\text{\#participating clients}}, 1\right\}$ layers. This means that if there are more trainable layers than participating clients, each client sends back weights for multiple layers.

**Per-Iteration Communication.** Unlike FEDSGD, where gradients are sent back to the server and aggregated after each iteration, SPRY offers additional communication cost savings. In this communication mode, SPRY only requires sending the `jvp` scalar value back to the server after each iteration of fine-tuning. Since the server has the seed value, it can generate the same random perturbation used by each client. Using the received `jvp` values and the generated random perturbations, the server can then compute the gradients and update the model weights.

A detailed breakdown of communication and computation costs is given in Appendix F.

## 4 Theoretical Analysis

This section theoretically analyzes convergence behaviors of SPRY. SPRY has a unique aggregation rule for the trainable weights as each client trains a subset of weights. It is also the first work to utilize forward gradients to train LLMs in FL settings where clients could have heterogeneous data distributions. Therefore, we theoretically analyze the effects of (a) data heterogeneity on gradient estimations of SPRY, and (b) configurations in SPRY, including the number of FL rounds, dimension of perturbations, the number of perturbations per iteration, data heterogeneity, and the number of participating clients, on the convergence of SPRY. The proofs are detailed in Appendix I.

**Theorem 4.1** (Estimation of the Global Gradient). *In SPRY, global forward gradients $\nabla\hat{f}$ of the trainable weights $w \in \mathbb{R}^d$, with the corresponding weight perturbations $v \in \mathbb{R}^d$, computed by $M$ participating clients is estimated in terms of true global gradients $\nabla f$ as,*

$$\mathbb{E}[\nabla\hat{f}(w,v;\mathcal{D})] = \nabla f(w) + \frac{1}{\widetilde{M}}\begin{bmatrix} \sum_{m\in\widetilde{\mathcal{M}}_1}\sum_{c=1}^{C}\alpha_{m,c}\mathbb{E}_{(x,y_c)\in\mathcal{D}}\left[\nabla\hat{f}_m(w_{[1,\frac{d}{M}]}, v_{[1,\frac{d}{M}]}; (x,y_c))\right] \\ \sum_{m\in\widetilde{\mathcal{M}}_2}\sum_{c=1}^{C}\alpha_{m,c}\mathbb{E}_{(x,y_c)\in\mathcal{D}}\left[\nabla\hat{f}_m(w_{[\frac{d}{M}+1,\frac{2d}{M}]}, v_{[\frac{d}{M}+1,\frac{2d}{M}]}; (x,y_c))\right] \\ \vdots \end{bmatrix}^T \quad (4)$$

*where the expectation is under the randomness of sampled data and random perturbation $v$. $C$ is total number of classes and $\alpha_{m,c} = \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)$. For a class $c$, $n_c$ is its sample count, $\alpha_c$ is its Dirichlet concentration parameter. For a client $m$, $n_{m,c}$ is the sample count of the $c^{th}$ class and $\mathcal{D}_m$ is the size of the data of client $m$. The global data is $\mathcal{D} = \sum_{m\in\mathcal{M}}\mathcal{D}_m$. $\widetilde{\mathcal{M}}$ is the set of clients training an arbitrary subset of weights, $\widetilde{M} = |\widetilde{\mathcal{M}}_i|, \forall i \in [M/d]$.*

**Discussion.** We focus on analyzing how data heterogeneity affects the estimation error between the global forward gradient and the global true gradient. Specifically, the estimation error of SPRY depends on the coefficient $\alpha_{m,c} = \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)$. In *data homogeneous* settings, where all clients share the same data distribution, the Dirichlet concentration parameter $\alpha_c$, is 1 for any class $c$. As the ratio of $n_c/|\mathcal{D}|$ (total samples in a class to total samples globally) matches the ratio of $n_{m,c}/|\mathcal{D}_m|$ (total samples in a class to total samples for a client $m$), the bias term becomes 0 since $\alpha_{m,c} = 0, \forall m, c$. Hence, the *global forward gradients are unbiased estimators* of the true global gradients. In *data heterogeneous* settings, when data across clients becomes more heterogeneous, $\alpha_c \to 0$ and thus $\alpha_{m,c} \to n_c/|\mathcal{D}|$, increasing the estimation error. Hence the global forward gradients are *biased estimators* that *depend on the distances of the data distributions* across participating clients.

**Theorem 4.2** (Convergence Analysis). *Under the assumptions on $L$-smoothness (Asmp I.1), bounded global variance $\sigma_g^2$ between global true gradients and aggregated expected gradients of each client*

*(Asmp I.2), and bound on gradient magnitude $G$ (Asmp I.3); and the following conditions on the local learning rate $\eta_\ell$,*

$$\eta_\ell = \min \left\{ \mathcal{O}\left(\frac{\tau^2}{\sqrt{\beta_2}\eta G L}\right)^{\frac{1}{2}}, \mathcal{O}\left(\frac{1}{\sqrt{\beta_2} G}\right), \mathcal{O}\left(\frac{\tau^3}{\sqrt{\beta_2(1-\beta_2)}G^2}\right)^{\frac{1}{2}}, \mathcal{O}\left(\frac{\widetilde{M}K}{\beta_2 G(3d+K-1)\sum_{m\in[M]}\sum_{c\in[C]}\alpha^2_{m,c}}\right)\right\}; \tag{5}$$

*The global true gradients of* Spry *satisfies the following bound,*

$$\min_{0\le r\le R} \mathbb{E}_r||\nabla f(w^{(r)})||^2 \le \frac{f(w^{(0)}) - \mathbb{E}_R[f(w^{(R)})]}{\eta R} + \left(2 + \frac{\eta\eta_\ell L}{2\tau^2} + \frac{\sqrt{1-\beta_2}G\eta_\ell}{\tau^3}\right)\left(\frac{\sigma_g^2(1-s)(3d+K-1)}{\widetilde{M}K}\right)\sum_{m\in\mathcal{M}}\sum_{c\in[C]}\alpha^2_{m,c}, \tag{6}$$

*where $R$ is the total number of FL rounds, $w \in \mathbb{R}^d$ are trainable weights, $v \in \mathbb{R}^d$ are random perturbation, $K$ is count of random perturbations per iteration, $\eta$ is global learning rate, $\tau$ is adaptability constant, and $s$ is client sampling rate. Rest of the symbols are defined in Theorem 4.1.*

**Discussion.** We focus on analyzing how different configurations in Spry affect its convergence. **(a)** *The number of FL rounds ($R$):* The upper bounds of the norm of the global forward gradient in Eq. 6 decrease in proportion to the inverse of $R$, indicating the convergence of the algorithm up to the limits of data heterogeneity. **(b)** *Dimension of perturbations ($d$):* $\eta_\ell \propto \frac{1}{d}$ shows that as the number of weights to be perturbed increases, the learning rate must decrease. A lower learning rate can make convergence slower, or worse, as our empirical experiments in Appendix G will show, not converge at all. **(c)** *The number of perturbations per iteration ($K$):* We observe $K$ both in nominator and denominator, which indicates that increasing $K$ brings little advantage in convergence speed. Results in Appendix G confirm the above statement. **(d)** *Data heterogeneity:* $\eta_\ell \propto \frac{1}{\alpha^2_{m,c}}$ shows that more homogeneous data distributions across clients allow higher learning rate, and hence faster error reduction. This observation is corroborated by the comparison of convergence speeds between homogeneous and heterogeneous clients in Appendix H. **(e)** *The number of clients training same subset of weights:* $\eta_\ell \propto \widetilde{M}$ shows that more clients training the same subset of weights is beneficial for faster convergence, similar observation is shown in Appendix G.

The theorem also sheds light on the accuracy performance gap between Forward-mode AD and backpropagation in FL settings. The upper bounds on the global forward gradient norm includes a second term that increases with $\alpha^2_{m,c}$ and variance $\sigma^2_g$, but does not decrease with $R$. This results in a gap between estimation errors of backpropagation-based methods like FedAvg and Spry.

## 5 Empirical Evaluation

We empirically evaluate Spry on 8 language tasks, 5 medium, and 3 large language models, under various FL settings. Our evaluation measures Spry's prediction performance, peak memory consumption, and time-to-convergence. We also ablate Spry's components to study the impact of communication frequency, the number of trainable parameters, the number of perturbations per iteration, the number of participating clients, and the importance of splitting layers on performance.

**Datasets, Tasks, and Models.** Our evaluation uses 8 datasets: **AG News** [31] (4-class classification), **SST2** [32] (2-class classification), **Yelp** [31] (2-class classification), **Yahoo** [31] (10-class classification), **SNLI** [33] (3-class classification), **MNLI** [34] (3-class classification), **SQuADv2** [35] (Closed-book question answering), and **MultiRC** [36] (2-class classification). We chose these datasets because they allow us to generate heterogeneous splits in FL settings using Dirichlet distribution [37]. The default dataset split is across 1,000 clients, except the smallest datasets **SST2** and **MultiRC**, where there are 100 clients. **SQuADv2** has 500 total clients. Each dataset has two versions: **(i)** Dirichlet $\alpha = 1.0$ (Homogeneous split), and **(ii)** Dirichlet $\alpha = 0.1$ (Heterogeneous split).

Our evaluation uses the following language models: **OPT13B** [38], **Llama2-7B** [21], **OPT6.7B** [38], **RoBERTa Large** (355M) [39], **BERT Large** (336M) [40], **BERT Base** (110M) [40], **DistilBERT Base** (67M) [41], and **Albert Large V2** (17.9M) [42]. For the billion-sized models, we use 4-bit quantization. For all the models, we use LoRA as the PEFT method. Appendix B describes the datasets and hyperparameters in more detail.

Table 1: Generalized accuracy for SPRY and its backpropagation- and zero-order-based counterparts on RoBERTa Large and LLMs. SQuADv2 uses F1 score. ↑ shows that higher values are better. The datasets are split with Dir $\alpha = 0.1$. ◇ = Llama2-7B. ⋆ = OPT6.7B. □ = OPT13B. SPRY outperforms the best-performing zero-order-based methods by 5.15–13.50% and approaches the performance of backpropagation-based methods, with a difference of 0.60–6.16%.

| | Backpropagation-based Methods ↑ | | Zero-order-based Methods ↑ | | | First-order Forward Mode AD ↑ | Difference between performances of SPRY and | |
|---|---|---|---|---|---|---|---|---|
| | FEDAVG | FEDYOGI | FWDLLM+ | FEDMEZO | BAFFLE+ | SPRY | best-performing backpropagation method ↑ | best-performing zero-order method ↑ |
| AG News | 93.07% | 92.77% | 76.94% | 70.56% | 57.69% | 87.89% | −5.18% | 10.95% |
| SST2 | 88.00% | 92.14% | 84.41% | 72.17% | 61.57% | 91.54% | −0.60% | 7.13% |
| SNLI | 86.45% | 79.31% | 74.30% | 69.57% | 62.10% | 82.66% | −3.79% | 8.36% |
| MNLI | 84.29% | 84.98% | 72.66% | 66.66% | 62.85% | 80.32% | −4.66% | 7.66% |
| Yahoo | 67.37% | 63.08% | 56.06% | 44.69% | 37.81% | 61.21% | −6.16% | 5.15% |
| Yelp | 90.48% | 79.10% | 71.83% | 65.10% | 55.99% | 85.33% | −5.15% | 13.50% |
| MultiRC ◇ | 47.56% | 72.53% | 64.58% | N/A | 58.12% | 68.65% | −3.88% | 4.07% |
| SQuADv2 ⋆ | 19.06 | 19.91 | 13.46 | 13.09 | 11.09 | 16.75 | −3.16 | 3.29 |
| SQuADv2 □ | 11.88 | 11.30 | 7.85 | 8.07 | 6.92 | 8.84 | −3.04 | 0.77 |

**Comparison Counterparts and Metrics.** We compare SPRY to (a) Backpropagation-based federated optimizers FEDAVG [1], FEDYOGI [25], FEDSGD (Variant of FEDAVG with per-iteration communication) [1], (b) Zero-order federated methods FEDMEZO (federated version of MEZO [18]), BAFFLE [20], FWDLLM [19], all based on finite difference. MEZO uses prompt-based finetuning to improve the performance of finite differences. FWDLLM generates a random perturbation that has a high cosine similarity to the global gradients of the previous rounds. BAFFLE generates ∼100–500 perturbations per iteration. More details of these methods are in Appendix A. The original implementations of FWDLLM and BAFFLE had excessive memory usage in their implementations. We improve their codebase to be memory-efficient by perturbing only the trainable weights, similar to SPRY. We refer to our implementation as FWDLLM+ and BAFFLE+.

Evaluation of SPRY and its counterparts for classification tasks is on **generalized accuracy** $Acc_g$ and **personalized accuracy** $Acc_p$, which are metrics measured on server-side aggregated model and client-side locally updated model, respectively. Similarly, for question-answering tasks, we measure **Exact Matches** and **F1 Score**. We also measure **time to convergence** and **peak memory consumption** during training. Our convergence criterion is the absence of change in the variance of a performance metric, assessed at intervals of 50 rounds.

SPRY is implemented in Flower [43] library. Quantization is done using AutoGPTQ [44]. For the zero-order methods, we used their respective client-side implementations with the server simulation structure of Flower. We utilized two Nvidia 1080ti to conduct all experiments of sub-billion sized models and billion-sized models for SPRY and its zero-order methods. We used two RTX8000s and two A100s for Llama2-7B and OPT models on backpropagation-based methods respectively. Each experiment was run thrice with 0, 1, and 2 as seeds.

## 5.1 Accuracy Performance Comparison

Table 1 reports the accuracy performance of SPRY and its backpropagation- and zero-order-based counterparts on heterogeneous datasets for million-sized RoBERTa Large, and billion-sized Llama2-7B, OPT6.7B, and OPT13B. Similar results on personalized performance is shown in Appendix H, Figure 5. Results on MultiRC for FEDMEZO are absent since the prompt-based finetuning variant of Llama2-7B was unavailable. Results on more model architectures and dataset combinations are available in Appendix G. Details on the learning curves, homogeneous dataset splits, and variance across 3 runs are in Appendix H.

Overall, SPRY achieves 5.15–13.50% higher generalized accuracy and 4.87–12.79% higher personalized accuracy over the best-performing zero-order-based methods across all datasets. FWDLLM+, the best-performing zero-order counterpart, attempts to reduce the effect of numeric instability of finite differences by (a) Sampling $K$ perturbations (default $K = 10$) per batch for each client and picking 1 perturbation per batch that has the highest cosine similarity with the previous round's aggregated gradients and (b) Only picking trained weights from clients whose computed gradients have variance lower than a set threshold. However, we posit that this strategy leads to some clients getting excluded due to a low variance threshold or outlying clients getting included due to a high variance threshold. Besides, picking new perturbations based on the previous round's aggregated gradients in the initial rounds can damage the learning trajectory. While BAFFLE+ samples more perturbation for each batch to make zero-order finite differences more tractable, the scale of language models demands perturbations on the scale of 500-1000 per batch, which becomes computationally infeasible. FEDMEZO manages to outperform BAFFLE+ due to its prompt-based finetuning trick but still falls short due to only using 1 perturbation per batch for finite differences on each client. In contrast, Forward-mode AD used in SPRY avoids the numerical instability from finite differences and improves accuracy by reducing trainable weights assigned to each client.

Compared to backpropagation-based methods, FEDAVG and FEDYOGI, SPRY manages to come as close as 0.60-6.16% of generalized accuracy and 2.50–14.12% of personalized accuracy. The performance gap between backpropagation and Forward-mode AD arises because in backpropagation, weight updates are more accurate as all gradients are computed directly using the error signal of the objective function. In contrast, Forward-mode AD relies on random perturbations, which is relatively less accurate for gradient estimation. Nonetheless, the advantages of SPRY become evident when we see the peak memory consumption of Forward-more AD compared to backpropagation, which we will discuss next.

## 5.2 Peak Memory Consumption Comparison

Figure 2 shows peak memory consumption of backpropagation (used in FEDAVG, FEDYOGI), zero-order finite differences (used in FWDLLM+, BAFFLE+, FEDMEZO), and first-order forward mode AD (used in SPRY). The methods are profiled for a single client.

Compared to backpropagation, Forward-mode AD reduces peak memory usage of RoBERTa Large by 27.90%, Llama2-7B by 81.73%, OPT6.7B by 86.26% and OPT13B by 85.93%. The sizes of trainable parameter (colored 🟥) and gradient + optimizer state (colored 🟦) are consistent across the 3 modes of computing gradients for all 4 models. Hence the savings come from the reduced memory footprint related to activations (colored 🟨) in Forward-mode AD. Compared to backpropagation-based methods, the memory cost related to activations is decreased by 12.12–49.25× in SPRY. Unlike storing all the intermediate activations in backpropagation, Forward-mode AD only has to store the previous layer's activation in a forward pass. The activation footprint of Forward-mode AD is equal to the size of the largest activation.

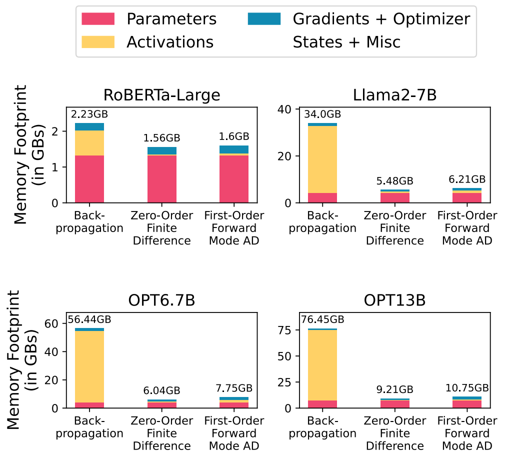


Figure 2: Peak memory consumption of SPRY's Forward-mode AD versus backpropgation- and zero-order-based methods. RoBERTa Large, Llama2-7B, and OPT6.7B are profiled with a batch size of 8, and OPT13B with a batch size of 4. SPRY reduces total memory usage by 27.90–86.26% compared to backpropagation-based methods. The 1.54–1.96× additional memory SPRY uses, compared to zero-order-based methods, is offset by the accuracy gains (§ 5.1).

Against zero-order methods, Forward-mode AD activations cost 1.96×, 1.95×, 1.83×, and 1.54× more for RoBERTa Large, Llama2-7B, OPT6.7B, and OPT13B respectively. The increasing cost comes from parallel evaluations of (a) the objective function on the original weights and (b) `jvp` computation on the perturbations in a single forward pass. However, as discussed in § 5.1, the increased memory cost is offset by a boost of up to 13.50% in accuracy performance. And as § 5.3 will discuss, Forward-mode AD reaches convergence faster than zero-order methods since it takes fewer steps to compute a closer gradient estimation.

## 5.3 Time to Convergence Comparison

Figure 3 shows wallclock time-to-convergence for SPRY and its counterparts. We observe that SPRY is 1.15-1.59×, 6.16-20.28×, and 1.34-2.98× faster than the zero-order methods FWDLLM+, BAFFLE+, and FEDMEZO respectively. For a client in each round, SPRY achieves a faster per-round computation time of 1.46×, 28.57×, and 1.80× on average, against FWDLLM+, BAFFLE+, and FEDMEZO. Forward-mode AD achieves faster convergence and faster per-round computation by providing a more accurate gradient estimation through a single perturbation per batch, leading to fewer steps needed to reach convergence. Since each client in SPRY only trains partial weights, it gains a speedup of 1.14× over backpropagation-based FEDAVG, FEDYOGI, and FEDSGD for RoBERTa Large. However, compared to the backpropagation-based methods, SPRY slows down for billion-sized LMs. We attribute this loss of speedup to the way `jvp` is computed in Forward-mode AD. `jvp` is computed column-wise, while its counterpart `vjp` in backpropagation are computed row-wise. The column-wise computation incurs time overhead.

### 5.4 Ablation Studies

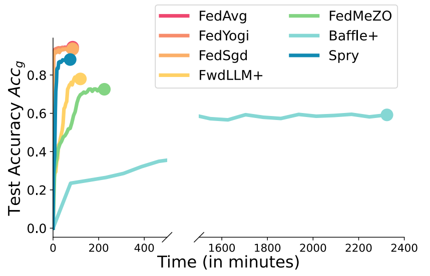
We summarize the ablation experiments on various components of SPRY. Further discussions are in Appendix G.

**SPRY can generalize to other language model architectures.** Similar to the observations from Table 1, we see a trend of SPRY outperforming the best performing zero-order method FWDLLM+ by 3.15-10.25% for generalized accuracy, demonstrating that SPRY can generalize to other language model architectures.

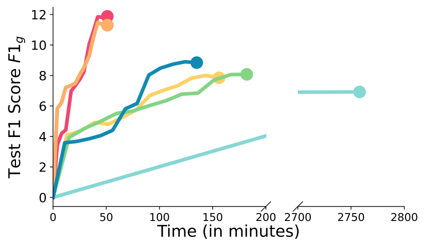
**SPRY is compatible with other PEFT methods.** We integrate SPRY with different PEFT methods like IA3, BITFIT, and CLASSIFIER-ONLY FINETUNING. Results shows that LORA with SPRY performs the best, with accuracy improvements of 10.60-16.53%.

**Effects of the number of trainable weights.** We change the number of trainable weights by controlling the rank $r$ and scale $\alpha$ hyperparameters in LORA. Results show that SPRY achieves the highest accuracy with ($r$=1, $\alpha$=1) setting, which has the smallest trainable weight count. The result is consistent with our theoretical analysis in §4.

**Effects of communication frequency.** Per-iteration communication in SPRY has been shown to boost accuracy by 4.47% compared to per-epoch communication. This improvement brings the accuracy within 0.92% and 0.96% of FEDAVG and FEDSGD, respectively.



(a) AG News with RoBERTa Large



(b) SQuADv2 with OPT13B

Figure 3: Time to convergence for SPRY and its counterparts. SPRY achieves faster convergence than zero-order methods due to more accurate gradient estimations in a single perturbation.

**Effects of perturbation count per batch.** We observe that increasing the number of perturbations per batch ($K$) for Forward-mode AD has little to no impact on the end prediction performance of SPRY, with $K = 100$ improving the generalized test accuracy by 1.1% over $K = 1$. The benefits of increasing $K$ are however seen in the convergence speed. Setting $K = 10$ achieves a steady state (of accuracy ~86%) around the 200$^{\text{th}}$ round, while the setting with $K = 1$ takes 500 rounds.

**Effects of participating client count.** Increasing the client count increases the prediction performance of SPRY. For the SST2 dataset, with the total client count fixed to 100, the three settings $C = 10$, $C = 50$, and $C = 100$ produce accuracies of 85.14%, 86.56%, and 88.08%, respectively. We also see an improvement in the convergence speed as the participating client count increases. To achieve an accuracy of ~85%; $C = 10$, $C = 50$, $C = 100$ require 500, 450, and 150 rounds, respectively.

**Importance of splitting weights.** To understand the effects of splitting, we conduct the following two experiments: (a) With FEDAVGSPLIT, we apply the strategy of splitting trainable layers across clients (see § 3.1) to backpropagation-based FEDAVG, and (b) With FEDFGD, we omit the splitting strategy of SPRY for FL with Forward-mode AD. We observe that FEDAVGSPLIT fails to achieve similar accuracy to FEDAVG with a drop of 2.60-10.00%. FEDFGD fails to converge as the size of trainable weights increases, e.g., with RoBERTa Large with LORA, that has 1.15M trainable weights. This proves the necessity of splitting trainable weights for Forward-mode AD in SPRY.

### 5.5 Communication and Computation Costs

Tables 2 and 3 in Appendix F shows the communication and computation overhead of SPRY against all its baselines. We summarize the main observations here:

**SPRY has lower communication cost due to the splitting strategy and scalar `jvp`.** Suppose $w_g$ is the total trainable parameter count of a model to be trained in federated setting. The backpropagation-based baselines; FEDAVG (per-epoch communication), FEDSGD (per-iteration communication), and FEDYOGI (per-epoch communication) transmit the entire set of trainable parameters to each participating client, and receives the same from each participating client. That results in "client to server" communication cost of $w_g$ and "server to client" communication cost of $w_g \times M$. The per-epoch versions of the zero-order baselines (FEDMEZO, BAFFLE, FWDLLM) follow a similar logic due to all the parameters needing to be transmitted to each client, with "client to server" communication costing $w_g$, and "server to client" communication taking the cost of $w_g \times M$. However, the per-iteration versions of the zero-order baselines fare better, with "client to server" communication only requiring each client sending a scalar finite difference (cost of 1), and "server to client" communication accruing $(w_g + 1) \times M$ cost, due to the server also needing to send a scalar seed to each client now.

Meanwhile, SPRY only needs to send $\max\left(\frac{L}{M}, 1\right)$ layers (where $L$ is the total layer count of a model, and $M$ is the participating count of clients for a round), each layer of size $w_\ell$. Hence, for per-epoch "client to server" communication accrues $w_\ell \max\left(\frac{L}{M}, 1\right)$, which is smaller than $w_g$. Similarly, per-epoch "server to client" communication costs $w_\ell M \max\left(\frac{L}{M}, 1\right)$, which is also smaller than the cost of $w_g M$ related to the baselines. For per-iteration communication, clients only need to send a scalar `jvp` (cost of 1) to the server; this matches the communication cost of per-iteration zero-order methods. Server needs to send a total of $w_\ell M \max(L, M)$ (derivation given in Table 2), which is a smaller cost than the costs of backpropagation and zero-order methods.

**SPRY accrues lower computation cost due to lower trainable parameter count.** SPRY's client-side computation cost is traded off by a faster convergence to higher accuracy through better gradient approximations compared to finite difference-based methods. And SPRY is the least computationally expensive on the server side due to needing to aggregate fewer parameters from the clients.

Let's assume that matrix multiplication costs $c$ for each layer, resulting in a forward pass cost of $c$. The cost of backpropagation is $2c$ because the computation of the current layer's weight gradient is $c$, and the cost of computing the previous layer's activation gradient is another $c$. `jvp` computation in SPRY takes additional cost of $c$ for each layer. Moreover, since `jvp` calculation happens through column-by-column vector multiplications, the related overhead is quantified by $v$.

Hence backpropagation-based methods FEDAVG, FEDSGD, and FEDYOGI computationally costs $3Lc$ at client, and costs $w_g(M-1)$ at server (due to additions). Note that $L$ amounts to all layers in the model here. FEDMEZO costs $L(2c + 3w_\ell)$ at client through two forward passes, and generating perturbations three times. FWDLLM and BAFFLE costs $KL(2c + w_\ell)$ due to $K$ perturbations for all $L$ layers, with two forward passes and generation perturbations once. Against that, SPRY costs $2\max(\frac{L}{M}, 1)(c + v) + w_\ell L$ for a smaller count of $L$, traded-off by the `jvp` computation cost of $v$.

On the server side, SPRY is the least computationally demanding. SPRY needs to aggregate a subset of layer weights from only the clients that were assigned to those layers. Computation cost on the server-side changes based on the communication frequency per-iteration communication incurs an additional overhead of $w_\ell L(\frac{M}{L} + 1)$ and $w_\ell L(M + 1)$ (generation of perturbations at the server-side, and multiplying those perturbations with aggregate of the `jvp` values received from the clients) for SPRY and its zero-order counterparts respectively.

## 6 Conclusion

SPRY enables finetuning medium and large language models in cross-device FL. It introduces a training strategy where trainable weights are split across federated clients, so each client only applies Forward-mode AD to a fraction of the weights. This approach significantly reduces the memory footprint compared to backpropagation and achieves better gradient estimation, resulting in higher accuracy and faster convergence than zero-order methods. Experiments on various language tasks and models demonstrate SPRY's effectiveness in reducing memory usage while maintaining accuracy comparable to backpropagation. We formally prove that the estimation bias of the global forward gradients in SPRY depends on data heterogeneity across clients. We also analyzed how the convergence rate of SPRY relates to the configurations of SPRY and FL settings including properties of weight perturbations, data heterogeneity, and the number of clients and FL rounds.

## Acknowledgments and Disclosure of Funding

This material is based upon work supported by the National Science Foundation under grant no. CCF-2210243, DMS-2220211, CNS-2224054, CNS-2312396, and CNS-2338512, and by Adobe.

## References

[1] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, Proceedings of Machine Learning Research. PMLR, 2017.

[2] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista A. Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaïd Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konecný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 2021.

[3] Tyler J Loftus, Matthew M Ruppert, Benjamin Shickel, Tezcan Ozrazgat-Baslanti, Jeremy A Balch, Philip A Efron, Jr. Gilbert R Upchurch, Parisa Rashidi, Christopher Tignanelli, Jiang Bian, and Azra Bihorac. Federated learning for preserving data privacy in collaborative healthcare research. *Digital Health*, 2022.

[4] Li Li, Xi Yu, Xuliang Cai, Xin He, and Yanhong Liu. Contract theory based incentive mechanism for federated learning in health crowdsensing. *IEEE Internet of Things Journal*, 2022.

[5] Suresh Dara, Ambedkar Kanapala, A. Ramesh Babu, Swetha Dhamercherala, Ankit Vidyarthi, and Ruchi Agarwal. Scalable federated-learning and internet-of-things enabled architecture for chest computer tomography image classification. *Computers and Electrical Engineering*, 2022.

[6] Fabio Pinelli, Gabriele Tolomei, and Giovanni Trappolini. Flirt: Federated learning for information retrieval. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2023.

[7] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish

Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report. *arXiv 2303.08774*, 2024.

[8] Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernandez Abrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan Botha, James Bradbury, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakanksha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Díaz, Nan Du, Ethan Dyer, Vlad Feinberg, Fangxiaoyu Feng, Vlad Fienber, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, Guy Gur-Ari, Steven Hand, Hadi Hashemi, Le Hou, Joshua Howland, Andrea Hu, Jeffrey Hui, Jeremy Hurwitz, Michael Isard, Abe Ittycheriah, Matthew Jagielski, Wenhao Jia, Kathleen Kenealy, Maxim Krikun, Sneha Kudugunta, Chang Lan, Katherine Lee, Benjamin Lee, Eric Li, Music Li, Wei Li, YaGuang Li, Jian Li, Hyeontaek Lim, Hanzhao Lin, Zhongtao Liu, Frederick Liu, Marcello Maggioni, Aroma Mahendru, Joshua Maynez, Vedant Misra, Maysam Moussalem, Zachary Nado, John Nham, Eric Ni, Andrew Nystrom, Alicia Parrish, Marie Pellat, Martin Polacek, Alex Polozov, Reiner Pope, Siyuan Qiao, Emily Reif, Bryan Richter, Parker Riley, Alex Castro Ros, Aurko Roy, Brennan Saeta, Rajkumar Samuel, Renee Shelby, Ambrose Slone, Daniel Smilkov, David R. So, Daniel Sohn, Simon Tokumine, Dasha Valter, Vijay Vasudevan, Kiran Vodrahalli, Xuezhi Wang, Pidong Wang, Zirui Wang, Tao Wang, John Wieting, Yuhuai Wu, Kelvin Xu, Yunhan Xu, Linting Xue, Pengcheng Yin, Jiahui Yu, Qiao Zhang, Steven Zheng, Ce Zheng, Weikang Zhou, Denny Zhou, Slav Petrov, and Yonghui Wu. Palm 2 technical report. *arXiv 2305.10403*, 2023.

[9] Bill Yuchen Lin, Chaoyang He, Zihang Ze, Hulin Wang, Yufen Hua, Christophe Dupuy, Rahul Gupta, Mahdi Soltanolkotabi, Xiang Ren, and Salman Avestimehr. FedNLP: Benchmarking federated learning methods for natural language processing tasks. In *Findings of the Association for Computational Linguistics: NAACL 2022*. Association for Computational Linguistics, 2022.

[10] Yuanyishu Tian, Yao Wan, Lingjuan Lyu, Dezhong Yao, Hai Jin, and Lichao Sun. Fedbert: When federated learning meets pre-training. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2022.

[11] Alexander Borzunov, Max Ryabinin, Artem Chumachenko, Dmitry Baranchuk, Tim Dettmers, Younes Belkada, Pavel Samygin, and Colin A Raffel. Distributed inference and fine-tuning of large language models over the internet. *Advances in Neural Information Processing Systems*, 2024.

[12] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[13] Dongqi Cai, Yaozong Wu, Shangguang Wang, and Mengwei Xu. Fedadapter: Efficient federated learning for mobile nlp. In *Proceedings of the ACM Turing Award Celebration Conference - China 2023*. Association for Computing Machinery, 2023.

[14] Zeju Qiu, Weiyang Liu, Haiwen Feng, Yuxuan Xue, Yao Feng, Zhen Liu, Dan Zhang, Adrian Weller, and Bernhard Schölkopf. Controlling text-to-image diffusion by orthogonal finetuning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[15] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In *Advances in Neural Information Processing Systems*, 2022.

[16] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, 2021.

[17] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[18] Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D. Lee, Danqi Chen, and Sanjeev Arora. Fine-tuning language models with just forward passes. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[19] Mengwei Xu, Dongqi Cai, Yaozong Wu, Xiang Li, and Shangguang Wang. Fwdllm: Efficient fedllm using forward gradient. *arXiv 2308.13894*, 2024.

[20] Haozhe Feng, Tianyu Pang, Chao Du, Wei Chen, Shuicheng Yan, and Min Lin. Does federated learning really need backpropagation? *arXiv 2301.12195*, 2023.

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

[22] C. H. Richardson. An introduction to the calculus of finite differences. by c.h. richardson pp. vi, 142. 28s. 1954. (van nostrand, new york; macmillan, london). *The Mathematical Gazette*, 39(330), 1955. doi: 10.2307/3608616.

[23] Atılım Güneş Baydin, Barak A. Pearlmutter, Don Syme, Frank Wood, and Philip Torr. Gradients without backpropagation. *arXiv 2202.08587*, 2022.

[24] Atılım Günes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *The Journal of Machine Learning Research*, 2017.

[25] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Hugh Brendan McMahan. Adaptive federated optimization. In *International Conference on Learning Representations*, 2021.

[26] Chandra Thapa, Pathum Chamikara Mahawaga Arachchige, Seyit Camtepe, and Lichao Sun. Splitfed: When federated learning meets split learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

[27] Renrui Zhang, Jiaming Han, Chris Liu, Aojun Zhou, Pan Lu, Hongsheng Li, Peng Gao, and Yu Qiao. LLaMA-adapter: Efficient fine-tuning of large language models with zero-initialized attention. In *The Twelfth International Conference on Learning Representations*, 2024.

[28] Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, 2022.

[29] Kunjal Panchal, Sunav Choudhary, Subrata Mitra, Koyel Mukherjee, Somdeb Sarkhel, Saayan Mitra, and Hui Guan. Flash: concept drift adaptation in federated learning. In *International Conference on Machine Learning*, pages 26931–26962. PMLR, 2023.

[30] Shanshan Wu, Tian Li, Zachary Charles, Yu Xiao, Ziyu Liu, Zheng Xu, and Virginia Smith. Motley: Benchmarking heterogeneity and personalization in federated learning. *arXiv 2206.09262*, 2022.

[31] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Neural Information Processing Systems*, 2015. Available at `https://huggingface.co/datasets/ag_news`, `https://huggingface.co/datasets/yelp_polarity`, `https://huggingface.co/datasets/yahoo_answers_topics`, Accessed on 15 May, 2024.

[32] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2013. Available at `https://huggingface.co/datasets/stanfordnlp/sst2`, Accessed on 15 May, 2024.

[33] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2015. Available at `https://huggingface.co/datasets/stanfordnlp/snli`, Accessed on 15 May, 2024.

[34] Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Association for Computational Linguistics, 2018. Available at `https://huggingface.co/datasets/SetFit/mnli`, Accessed on 15 May, 2024.

[35] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, 2018. Available at `https://huggingface.co/datasets/rajpurkar/squad_v2`, Accessed on 15 May, 2024.

[36] Daniel Khashabi, Snigdha Chaturvedi, Michael Roth, Shyam Upadhyay, and Dan Roth. Looking beyond the surface: A challenge set for reading comprehension over multiple sentences. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Association for Computational Linguistics, 2018. Available at `https://huggingface.co/datasets/mtc/multirc`, Accessed on 15 May, 2024.

[37] Kunjal Panchal, Sunav Choudhary, Nisarg Parikh, Lijun Zhang, and Hui Guan. Flow: Per-instance personalized federated learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[38] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models. *arXiv 2205.01068*, 2022.

[39] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *arxiv 1907.11692*, 2019.

[40] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *arXiv 1810.04805*, 2018.

[41] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv 1910.01108*, 2019.

[42] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. *arXiv 1909.11942*, 2019.

[43] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, and Nicholas D Lane. Flower: A friendly federated learning research framework. *arXiv preprint 2007.14390*, 2020.

[44] AutoGPTQ, 2024. URL `https://github.com/AutoGPTQ/AutoGPTQ`.

[45] Richard L. Burden and J. Douglas. Faires. *Numerical analysis / Richard L. Burden, J. Douglas Faires.* Thomson Brooks/Cole, 8th ed. edition, 2005. ISBN 0534392008.

[46] Károly Jordán. *Calculus of finite differences*. American Mathematical Soc., 1965.

[47] Wenzhi Fang, Ziyi Yu, Yuning Jiang, Yuanming Shi, Colin N. Jones, and Yong Zhou. Communication-efficient stochastic zeroth-order optimization for federated learning. *IEEE Transactions on Signal Processing*, 2022.

[48] Rui Ye, Wenhao Wang, Jingyi Chai, Dihan Li, Zexi Li, Yinda Xu, Yaxin Du, Yanfeng Wang, and Siheng Chen. Openfedllm: Training large language models on decentralized private data via federated learning. *arXiv 2402.06954*, 2024.

[49] Tao Fan, Yan Kang, Guoqiang Ma, Weijing Chen, Wenbin Wei, Lixin Fan, and Qiang Yang. Fate-llm: A industrial grade federated learning framework for large language models. *arXiv 2310.10049*, 2023.

[50] Fan Lai, Yinwei Dai, Sanjay S. Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. Fedscale: Benchmarking model and system performance of federated learning at scale. *arXiv 2105.11367*, 2022.

[51] Jae Ro, Theresa Breiner, Lara McConnaughey, Mingqing Chen, Ananda Suresh, Shankar Kumar, and Rajiv Mathews. Scaling language model size in cross-device federated learning. In *Proceedings of the First Workshop on Federated Learning for Natural Language Processing (FL4NLP 2022)*. Association for Computational Linguistics, 2022.

[52] Shubham Malaviya, Manish Shukla, and Sachin Lodha. Reducing communication overhead in federated learning for pre-trained language models using parameter-efficient finetuning. In *Conference on Lifelong Learning Agents*. PMLR, 2023.

[53] Zhuo Zhang, Yuanhang Yang, Yong Dai, Qifan Wang, Yue Yu, Lizhen Qu, and Zenglin Xu. FedPETuning: When federated learning meets the parameter-efficient tuning methods of pre-trained language models. In *Findings of the Association for Computational Linguistics: ACL 2023*. Association for Computational Linguistics, 2023.

[54] Seonghwan Park, Dahun Shin, Jinseok Chung, and Namhoon Lee. Fedfwd: Federated learning without backpropagation. *arXiv 2309.01150*, 2023.

[55] Geoffrey Hinton. The forward-forward algorithm: Some preliminary investigations. *arXiv 2212.13345*, 2022.

[56] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 2023.

[57] Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv 2403.14608*, 2024.

[58] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

## A Related Work

SPRY uses first-order forward gradients for federated finetuning of language models. Hence, we review the literature on estimating gradients with low memory consumption and show how SPRY represents a significant advancement in finetuning language models in FL.

**Zero-order Gradients.** Gradients derived from finite difference [45, 22, 46] methods are called *zero-order gradients* since they don't involve Taylor expansion of the objective function $f$. MEZO [18] has shown that finite difference with one perturbation per batch does not reach convergence on its own without additional tricks like prompt-based finetuning, which are highly specific to the tasks. For a more accurate gradient approximation, an average of zero-order gradients derived from multiple ($\sim$10 to 100) random perturbations on the same input batch is required [24], leading to slow convergence. BAFFLE [20] and FEDZO [47] utilize zero-order gradients in federated settings. To train vision models (of parameter count $\leq$ 13M), BAFFLE requires (a) $\sim$100-500 perturbations per batch for each client respectively, and (b) per-iteration communication among clients like FEDSGD [1]. FEDZO also requires $\sim$20 perturbations per batch for a vision model of size $\leq$ 25M. Besides, numerical errors associated with finite difference make MEZO, BAFFLE, and FEDZO suffer from sub-optimal predictions compared to backpropagation-based counterparts. SPRY, using Forward-mode AD, computes gradients more accurately in a single forward pass compared to averaged gradients obtained through finite difference methods. This higher accuracy is achieved without needing modifications to model architectures or task structures, while also maintaining a memory footprint similar to that of finite difference methods.

**First-order Forward Gradients.** Gradients derived from Forward-mode Auto-Differentiation (AD) are considered first-order since it involves computing partial derivatives of the intermediate activations with respect to the input. We guide interested readers to the survey on different modes of automatic differentiation [24]. FGD [23] shows preliminary results on the speedup and comparable accuracy achieved by Forward-mode AD against backpropagation. The challenge that makes Forward-mode AD less popular than backpropagation is that gradients derived from forward mode require more `jvp` column-by-column evaluations per input batch as the number of trainable weights increases. Moreover, evaluation of FGD is limited to a multi-layer perceptron of size $d \approx$1.8M. Direct use of FGD to finetune language models leads to slow or no convergence. SPRY splits the trainable layers of a large language model across multiple clients in FL, letting each client finetune only a small subset of weights through forward gradients.

**Training or Finetuning Language Models in Federated Learning.** In recent years, several frameworks have been proposed to train or finetune LLMs in FL [48–50]. The backpropagation-based methods [10, 51], even with parameter efficient finetuning (PEFT) and quantization [13, 52, 53], have large memory footprints due to the overhead related to activations, gradients, and gradient history storage for adaptive optimizers [18].

FEDFWD [54] applies FWDFWD [55] (which measures "goodness" of forward pass activations to judge which perturbations are useful) in FL, but FWDFWD struggles as model size scales up. FWDLLM uses zero-order gradients to finetune language models. It samples $\sim$10 perturbations per batch. For each batch, it picks 1 perturbation that has the highest cosine similarity with the previous round's gradients. Sampling new perturbations based on aggregated gradients from previous rounds during the initial stages can disrupt the learning trajectory. SPRY requires 1 perturbation (without resampling) per batch to reach a higher prediction performance faster than FWDLLM.

## B Datasets and Hyperparameters

Here we provide details of the datasets and their corresponding training hyperparameters used in this work.

**Simulating Heterogeneity through Dirichlet Distribution.** For each of the tasks, the class distribution each client gets depends on the Dirichlet distribution, where a parameter $\alpha$ regulates the concentration of samples of a particular class for a specific client. Dir $\alpha = 1.0$ means all clients have homogeneous datasets where each class is equally likely to be on each client. With Dir $\alpha \rightarrow 0$, the datasets of each client get more heterogeneous, where the sample distribution of each class is more likely to be concentrated on only a subset of clients.

**Default Hyperparameters.** Unless otherwise mentioned in dataset-specific paragraphs, the default hyperparameters for each method and for all datasets are stated here. For the backpropagation-based methods FEDAVG, FEDYOGI, and FEDSGD; we will fix the number of **epochs** to 1 since the goal of this work is to inch closer to backpropagation-like prediction performance while reducing the memory footprint. All the experiments have been run for 1500 **FL rounds**, except the experiments on OPT models, which are run for 600 FL rounds. Our observation from hyperparameter-tuning shows that the learning rate that gives the best performance is the same for all studied methods. BAFFLE and its memory-efficient improvement BAFFLE+ made by us, can perform better as the number of perturbations per batch increases, but due to the scale of experiments with 10-100 per round and up to 1500 rounds in the FL setting, we limit the total perturbations per batch of BAFFLE+ to 20 perturbations per batch and fixed finite difference step size $\sigma$ = 1e-4. FWDLLM+ samples 10 perturbations for each batch, finite difference step size of $\sigma$ = 1e-2. FEDMEZO samples 1 perturbation for each batch, with finite difference step size of $\sigma$ = 1e-3. FEDMEZO also requires 3-5 epochs for each client. SPRY has 1 perturbation per batch for each client. For SPRY and its zero-order counterparts (BAFFLE+, FWDLLM+, and FEDMEZO), perturbations are **sampled for a normal distribution** with 0 mean and 1 variance. Default LORA $r$ and $\alpha$ are 1 and 1, respectively. All methods use ADAMW as client-side optimizer. Besides FEDAVG, all methods use FEDYOGI as **server-side optimizer**.

**AG News.** AG News dataset [31] has been derived from a corpus of 496,835 categorized news articles. A subset of the corpus is used that has 120,000 total training samples and 7,600 total testing samples spread equally across 4 classes. The news articles are classified into 4 classes: World, Sports, Business, and Sci/Tech. We split this data across 1000 clients. Each client gets an equal number of samples for train and test datasets. This dataset is under Creative Commons CCZero(CC0) public domain dedication.

Learning rate for backpropagation-based (FEDAVG, FEDYOGI, and FEDSGD), zero-order-based (FWDLLM, BAFFLE, and FEDMEZO), and first-order-based SPRY is {1e-3, 5e-4, **1e-4**, 1e-5}. The batch size is set to 8. The max sequence length is 128. All methods use ADAMW as a client-side optimizer, while SPRY performs better with SGD. Variance threshold of FWDLLM+ is 1e+1.

**SST2.** Stanford Sentiment Treebank Binary[32] (or SST2) dataset is for a binary sentiment classification task. The dataset has 11,855 sentences derived from a set of movie reviews. The corpus was parsed using the Stanford Parser into 215,154 discrete phrases annotated by 3 human judges. This dataset contains 67,349 training samples, 872 validation samples, and 1821 testing samples. This sentiment classification dataset has the following sentiments as classes: Positive, and Negative. These samples are equally split between 100 clients, depending on the Dirichlet distribution. This dataset is under Creative Commons CCZero public domain dedication.

Learning rate for backpropagation-based (FEDAVG, FEDYOGI, and FEDSGD), zero-order-based (FWDLLM, BAFFLE, and FEDMEZO), and first-order-based SPRY is {1e-3, 5e-4, **1e-4**, 1e-5}. The batch size is set to 8. The max sequence length is 64. Variance threshold of FWDLLM+ is 5e+0.

**Yelp.** The Yelp reviews dataset[31] is a binary classification dataset gathered during the 2015 Yelp Dataset Challenge. The full dataset has 1,569,264 total samples, and it defines 2 classification tasks. We use the polarity classification task, which is a binary classification problem. It considers 1-2 stars negative polarity and 3-4 stars positive polarity. This dataset has 280,000 training samples and 19,000 test samples in each polarity. We split this data into 1000 clients. This dataset is under the Apache License, Version 2.0.

Learning rate for backpropagation-based (FEDAVG, FEDYOGI, and FEDSGD), zero-order-based (FWDLLM, BAFFLE, and FEDMEZO), and first-order-based SPRY is {1e-3, 5e-4, 1e-4, **5e-5**, 1e-5}. The batch size is set to 8. The max sequence length is 128. Variance threshold of FWDLLM+ is 5e+1.

**Yahoo.** Yahoo! Answers Comprehensive Questions and Answers dataset [31] was gathered via the Yahoo! webscope program. The corpus itself contains 4,483,032 question/answer pairs, which were then formulated as a 10-class classification task. Each class in this dataset has 140,000 training and 5,000 testing samples.are The question/answer pairs are split into the following classes: 1. Society & Culture, 2. Science & Mathematics, 3. Health, 4. Education & Reference, 5. Computers & Internet, 6. Sports, 7. Business & Finance, 8. Entertainment & Music, 9. Family & Relationships, 10. Politics & Government. We split this dataset between 1000 clients, where each client gets an equal amount of data samples, where the data distribution is set by changing the Dirichlet distribution $\alpha$ to range from most homogeneous ($\alpha = 1$) to least homogeneous ($\alpha = 0$). This dataset is under the Apache License, Version 2.0.

Learning rate for backpropagation-based (FEDAVG, FEDYOGI, and FEDSGD), zero-order-based (FWDLLM, BAFFLE, and FEDMEZO), and first-order-based SPRY is {1e-3, 5e-4, **1e-4**, 1e-5}. The batch size is set to 8. The max sequence length is 128. Variance threshold of FWDLLM+ is 5e+1.

**SNLI.** The Stanford Natural Language Inference corpus [33] has 570,152 total sentence pairs. It is a natural language inference dataset, where the task is identifying if one sentence infers another. It has 550,152 training samples, 10,000 testing samples, and 10,000 evaluation samples. This dataset is split among 1,000 clients. The dataset has 3 classes: 1) The first sentence entails the second sentence, 2) The first sentence is neutral to the second sentence and 3) The first sentence contradicts the second sentence. This dataset is under CC BY-SA 4.0.

Learning rate for backpropagation-based (FEDAVG, FEDYOGI, and FEDSGD), zero-order-based (FWDLLM, BAFFLE, and FEDMEZO), and first-order-based SPRY is {1e-3, 5e-4, 1e-4, **5e-5**, 1e-5}. The batch size is set to 8. The max sequence length is 80. Variance threshold of FWDLLM+ is 1e+2.

**MNLI.** The Multi-Genre Natural Language Inference (MNLI) [34] corpus contains 432,702 sentence pairs which were crowd-sourced and then annotated with textual entailment information. This dataset is also a Natural Language Inference dataset as with SNLI. This dataset has 392,702 training, 20,000 evaluation, and 20,000 testing samples. These samples are split among 1,000 clients. This dataset draws from multiple sources, most of which are under the Open American National Corpus (OANC) license. The rest are under either the CC BY 3.0 Unported licenses or the CC BY-SA 3.0 licenses.

Learning rate for backpropagation-based (FEDAVG, FEDYOGI, and FEDSGD), zero-order-based (FWDLLM, BAFFLE, and FEDMEZO), and first-order-based SPRY is {1e-3, 5e-4, 1e-4, **5e-5**, 1e-5}. The batch size is set to 8. The max sequence length is 80. Variance threshold of FWDLLM+ is 1e+2.

**SQuADv2.** The Stanford Question Answering Dataset (SQuADv2) [35] consists of crowd-sourced questions about a set of Wikipedia articles. It is a reading comprehension dataset, where the answer to a question is a section (or a span) from the passage. It is also possible for the question to be unanswerable. The dataset contains 100,000 answerable and 50,000 unanswerable questions. This dataset was split into 500 clients. The heterogeneity is generated based on the topic labels (or "titles") associated with each question in the dataset. There are 35 titles available. This dataset is under the CC BY-SA 4.0 license.

Learning rate for backpropagation-based (FEDAVG, FEDYOGI, and FEDSGD), zero-order-based (FWDLLM, BAFFLE, and FEDMEZO), and first-order-based SPRY is {1e-3, 5e-4, 1e-4, **1e-5**}. The batch size is set to 8 for Forward-mode AD and zero-order differentiation-based methods. Backpropagation-based methods use a batch size of 4. The max sequence length is 400. Variance threshold of FWDLLM+ is 5e+1.

**MultiRC.** Multi-sentence Reading Comprehension (MultiRC) corpus is a dataset that contains short paragraphs with questions, whose answers can be found in the paragraph itself. We consider a task where we classify if the input question given as input is correct or false. It has 6k multi-sentence questions about 800+ paragraphs. Due to the scale of the dataset, we split it into 100 clients. This dataset is under the MIT license.

Learning rate for backpropagation-based (FEDAVG, FEDYOGI, and FEDSGD), zero-order-based (FWDLLM, BAFFLE, and FEDMEZO), and first-order-based SPRY is {1e-3, 5e-4, **1e-4**, 1e-5}. The batch size is set to 8. The max sequence length is 256. Variance threshold of FWDLLM+ is 1e+2.

## C Limitations and Future Work

SPRY achieves a remarkable drop in memory consumption due to Forward-mode AD not having to store activations during the forward pass. However, the current implementation of Forward-mode AD by PyTorch is still suboptimal in terms of computation time. The high computation time is attributed to the column-by-column computation of the intermediate results of `jvp`. Improving the computation of `jvp` such that it takes significantly less time (almost half) than backpropagation remains an open and interesting problem. Moreover, there's room for improvement in reducing the memory consumption to that of zero-order methods. In zero-order methods, the weights are perturbed and a forward pass is computed on the perturbed weights. However, with the current implementation of Forward-mode AD, perturbations create a separate copy from the original weights, which accrues additional overhead. To further utilize the computation capacity of clients in FL, device-heterogeneity-aware strategies on splitting and mapping layers to clients can be explored for SPRY, e.g., layer selection could happen on the client-side according to their data distributions.

## D Broader Impact

Through SPRY, we aspire to bring impact in terms of data privacy and accessible finetuning of large language models (LLMs) on edge devices.

For small and medium organizations and individual users, the cost to finetune these language models can be prohibitively expensive as the size of the trainable weights increases. With SPRY, we enable finetuning LLMs on resource-constrained edge devices with low memory consumption, making it feasible for a wider range of users. Moreover, the federated setting also provides benefits of data privacy. This can be ideal for use cases where LLMs can bring improved performance for a plethora of personalized downstream tasks, but the finetuning data containing sensitive and confidential information is never shared with a third party. With SPRY, such data remains on the user's device, ensuring privacy while still allowing for effective finetuning of the LLM.

However, making LLM finetuning accessible to a broader range of organizations and individuals comes with its own challenges like a spread of biases or misinformation. Without preprocessing and filtering client data on their devices, the LLMs can be fed harmful and misleading information. Hence, it is necessary to develop guardrails on what kind of data should be filtered in, to finetune LLMs with crowd-sourced compute resources.

## E Spry Pseudocode

Algorithm 1 shows the workflow of Spry.

**Algorithm 1:** Spry

```
Input: $R$: Total number of rounds, $r \in [R]$: Round index, $M$: Number of clients per round, $m \in [M]$:
       Client index, $\mathcal{M}$: Set of available clients, $s$: Client sampling rate, $\mathcal{D}_m$: Dataset of client $m$, $\eta_\ell$:
       Local learning rate, $\boldsymbol{w}^{(r)}$: Model weights at $r^{th}$ round, $\overline{\boldsymbol{w}}_m^{(r)}$: Subset of assigned trainable weights
       for client $m$ at $r^{th}$ round, $f$: Objective function.
Output: $\boldsymbol{w}^{(R+1)}$: Model at the end of the training
1  Main Spry
2      Server loads pre-trained LM $\boldsymbol{w}^{(1)}$ with PEFT
3      for $r \in [R]$ round do
4          Sample $M$ clients from $\mathcal{M}$ at rate of $s$
5          $\mathcal{L} \leftarrow$ list of trainable PEFT param names
6          client_layer_mapping $\leftarrow$ MapLayersToClients($\mathcal{L}$, $M$)
7          for $m \in [M]$ in parallel do
8              $\overline{\boldsymbol{w}}_m^{(r)} \leftarrow$ ClientTrain($\boldsymbol{w}^{(r)}$, client_layer_mapping[$m$])
9          end
10         Build $\boldsymbol{w}'^{(r)}$ with $\overline{\boldsymbol{w}}_m^{(r)}$ using client_layer_mapping[$m$], $\forall m \in [M]$
11         Use adaptive optimizer like FedYogi to build $\boldsymbol{w}^{(r+1)}$ based on the aggregated $\boldsymbol{w}'^{(r)}$
12     end
13 end
14 Function MapLayersToClients($\mathcal{L}$, $M$)
15     client_idx $\leftarrow$ 0;  mapping $\leftarrow$ {}
16     for name $\in \mathcal{L}$ do
17         rollover_idx $\leftarrow$ client_idx % $M$
18         mapping[rollover_idx].update(name)
19         client_idx $\leftarrow$ client_idx + 1
20     end
21     return mapping
22 end
23 Function ClientTrain($\boldsymbol{w}_m^{(r)}$, trainable_layers)
24     Freeze $\boldsymbol{w}_m^{(r)}$ parameters $\notin$ trainable_layers
25     Generate perturbations $\boldsymbol{v} \leftarrow \mathcal{N}(0, \mathbf{I}_{\overline{\boldsymbol{w}}_m^{(r)}.\text{shape}})$;
           $\forall$ trainable $\overline{\boldsymbol{w}}_m^{(r)}$
26     jvp $\leftarrow f$.ForwardAD($\overline{\boldsymbol{w}}_m^{(r)}, \boldsymbol{v}; \mathcal{D}_m$)
27     $\overline{\boldsymbol{w}}_m^{(r)} \leftarrow \overline{\boldsymbol{w}}_m^{(r)} - \eta_\ell(\boldsymbol{v} \cdot \text{jvp})$
28     return all trainable $\overline{\boldsymbol{w}}_m^{(r)}$
29 end
```

The function MapLayersToClients on Line 14 in Algorithm 1 shows how we have assigned only a few trainable layers to each client in FL, to make Forward-mode AD more effective at generating better estimation of the gradients. In function ClientTrain on Line 23 of Algorithm 1; each client $m$ gets a copy of the entire language model $\boldsymbol{w}_m^{(r)}$ and a list `trainable_layers` of parameter names the client $m$ has to train. A client $m$ freezes the parameters that are not included in its `trainable_layers`. And for each of the parameter $\overline{w}_m^{(r)}$ which need to be trained, Spry generates a corresponding random perturbation $v$ using a normal distribution $\mathcal{N}(0, \mathbf{I}_{\overline{w}_m^{(r)}.\text{shape}})$.

Once a client $m$ has obtained forward gradients of all the trainable parameters $\overline{\boldsymbol{w}}_m^{(r)}$, those parameters are locally updated with optimizers like SGD or Adam. The updated trainable parameters $\overline{\boldsymbol{w}}_m^{(r)}$ are sent back to the server. The server has a mapping of parameter names to client IDs, and hence it builds $\boldsymbol{w}'^{(r)}$ by using $\overline{\boldsymbol{w}}_m^{(r)}$ $\forall m \in [M]$. If there are multiple clients mapped to the same parameter, then we take a weighted average (similar to FedAvg) of all the parameters to build $\boldsymbol{w}'^{(r)}$. Spry uses adaptive optimizers like FedYogi at server-side on effective gradients $\Delta = \boldsymbol{w}'^{(r)} - \boldsymbol{w}^{(r)}$ to reduce the noise of forward gradients.

## F Communication and Computation Costs

### F.1 Communication Costs

Table 2 illustrates communication costs of SPRY and its backpropagation- and finite difference- based baselines. A discussion on communication modes of SPRY is also given in Section 3.2, "Per-Epoch Communication" and "Per-Iteration Communication".

Table 2: Communication cost of SPRY and all its baselines. $M$ is the count of participation clients. Total count of trainable parameters of a global model is $w_g = w_\ell L$ (for simplicity, we assume that each layer has $w_\ell$ parameters).

| Gradient computation | Method (Comm. frequency) | Communication cost (in parameter count) **from each client to server** for each communication round | Communication cost (in parameter count) **from server to all clients** for each communication round |
|---|---|---|---|
| Backpropagation (First-order gradients) | FEDAVG / FEDYOGI (Per-epoch) | $w_g$ | $w_g \times M$ |
| | FEDSGD (Per-iteration) | $w_g$ | $w_g \times M$ |
| Finite differences (Zero-order gradients) | FEDMEZO / FWDLLM / BAFFLE (Per-epoch) | $w_g$ | $w_g \times M$ |
| | FEDMEZO / FWDLLM BAFFLE (Per-iteration) | 1 (of finite difference scalar) | $(w_g + 1) \times M$ ("1" is for perturbation seed) |
| Forward-mode AD (First-order gradients) | SPRY (Per-epoch) | $w_\ell \times \max(L/M, 1)$ (Assuming $L\%M = 0$ for each of exposition) | $w_\ell \times \max(L/M, 1) \times M$ = $w_\ell \times \max(L, M)$ |
| | SPRY (Per-iteration) | 1 (of `jvp` scalar) | $(w_\ell \times \max(L/M, 1) \times M)$ $+(1 \times M)$ $= w_\ell \times \max(L, M) + M$ |

Here we discuss the costs related to those communication modes:

**Per-epoch Communication.** SPRY's client-to-server communication cost does not scale linearly with clients like in its backpropagation and finite-difference counterparts, but instead decreases or stays constant for $L$ as more clients are present. Server-to-client communication cost is lower in SPRY due to only sending one layer per client when $M > L$ or $\frac{L}{M}$ layers per client otherwise. This result follows from the below observation:

Backpropagation-based and finite-difference-based methods have a communication cost of $w_g$, where $w_g$ represents the global model size. Each client in $[M]$ (set of participating clients) receives all trainable parameters from the server, requiring the server to send a total of $w_g \times M$ parameters each round.

SPRY's communication cost per epoch is $w_\ell \max(\frac{L}{M}, 1)$, where $L$ is the layer count and $w_\ell$ is the count of parameters for each layer. Each client sends a subset of trainable parameters, incurring a communication cost of $\frac{w_\ell L}{M}$ parameters for $L > M$, and $w_\ell$ for $L \leq M$. When $L \leq M$, each client gets 1 layer, hence the communication for each client is $w_\ell$.

**Per-iteration Communication.** SPRY accrues lower communication cost than the finite difference and backpropagation counterparts due to the layer splitting strategy, and the server's ability to compute gradients based on the `jvp` value. This is because:

The communication cost from client to server for forward-mode AD and finite differences is 1. This is due to an FL round that involves (1) server selecting a random seed, (2) server sending it with trainable parameters to clients, (3) clients generating perturbations based on the seed, (4) deriving and sending back a scalar or finite difference scalar to the server, and then (5) server computing gradients by multiplying the derived perturbations with the seed.

The server to client communication is $(w_g + 1) \times M$, where the "+1" is due the randomness seed.

### F.2 Computation Costs

Table 3 shows the computation costs of SPRY and its baselines, where the client-side cost is for each iteration, and the server-side cost is for each round.

Briefly, SPRY's client-side computation cost is traded off by a faster convergence to higher accuracy through better gradient approximations compared to finite difference-based methods. Furthermore, SPRY is the least computationally expensive on the server side due to needing to aggregate fewer parameters from the clients.

Table 3 assumes that matrix multiplication costs $c$ for each layer, resulting in a forward pass cost of $c$. The cost of backpropagation is $2c$ because the computation of the current layer's weight gradient is $c$, and the cost of computing the previous layer's activation gradient is another $c$. `jvp` computation in SPRY takes additional cost of $c$ for each layer. Moreover, since `jvp` calculation happens through column-by-column vector multiplications (Sec 3.1 of [24]), the related overhead is quantified by $v$.

Table 3: Computation cost of SPRY and all its baselines. The client-side cost is for each iteration, and the server-side cost is for each round. $L$ is the layer count, $M$ is the participating client count, $c$ is the cost of matrix multiplication for each layer. $v$ is the overhead related to column-by-column vector multiplications of `jvp`. $w_\ell$ is the size of each layer and hence size of each layer's perturbation too. $K$ is the perturbation count per iteration. $K = 1$ for SPRY and FEDMEZO, and $\sim 20$ for BAFFLE and FWDLLM.

| Gradient computation | Method (Comm. frequency) | Computation cost of **each client for each iteration** | Computation cost of **the server for each round** |
|---|---|---|---|
| Backpropagation (First-order gradients) | FEDAVG / FEDYOGI (Per-epoch) | $3Lc$ | (Aggregating $L$ layer weights from $M$ clients) $(M-1) \times w_\ell L$ |
| | FEDSGD (Per-iteration) | $3Lc$ | $(M-1) \times w_\ell L$ |
| Finite differences (Zero-order gradients) | FEDMEZO (Per-epoch) | $L(2c + 3w_\ell)$ | $(M-1) \times w_\ell L$ |
| | FWDLLM / BAFFLE (Per-epoch) | $KL(2c + w_\ell)$ | $(M-1) \times w_\ell L$ |
| | FEDMEZO (Per-iteration) | $L(2c + 3w_\ell)$ | $w_\ell L + w_\ell ML + w_\ell(M-1)L = 2Mw_\ell L$ (perturbation generation + gradient calculation + weight update) |
| | FWDLLM / BAFFLE (Per-iteration) | $KL(2c + w_\ell)$ | $2Mw_\ell L$ |
| Forward-mode AD (First-order gradients) | SPRY (Per-epoch) | $2 \times \max(\frac{L}{M}, 1) \times (c+v) + w_\ell L$ | $\sum_{\mathcal{M} \subset [M]} \left((\lvert\mathcal{M}\rvert - 1) w_\ell \max(\frac{L}{M}, 1)\right)$ (usually, $\lvert\mathcal{M}\rvert = \max(\frac{M}{L}, 1)$) |
| | SPRY (Per-iteration) | $2 \times \max(\frac{L}{M}, 1) \times (c+v) + w_\ell L$ | $\sum_{\mathcal{M} \subset [M]} \left(2\lvert\mathcal{M}\rvert w_\ell \max(\frac{L}{M}, 1)\right)$ |

**Client-side per-iteration computation cost.** Backpropagation needs 3 matrix multiplication operations per layer. For zero-order methods, there are 2 matrix multiplications (incurred due to two forward passes) per layer, and per perturbation within a training iteration; and additional overhead $w_\ell KL$ for perturbation generation. MEZO requires generation of perturbations thrice for the same seed (Algorithm 1 in MEZO [18]).

SPRY 's computation cost is $2 \times \max(\frac{L}{M}, 1) \times (c + v) + w_\ell L$. Since SPRY allocates at most $\frac{L}{M}$ layers to each client, the computation cost only scales with $\max(\frac{L}{M}, 1)$, against its counterparts scaling with $L$. However, forward-mode AD computes `jvp` column-wise, while its counterpart `vjp` in backpropagation is computed row-wise. This results in time overhead ( ) if the number of trainable parameters exceeds the output size (1 as loss is scalar), which is the case for neural networks. Therefore, SPRY's per-iteration computation cost is higher compared to other approaches.

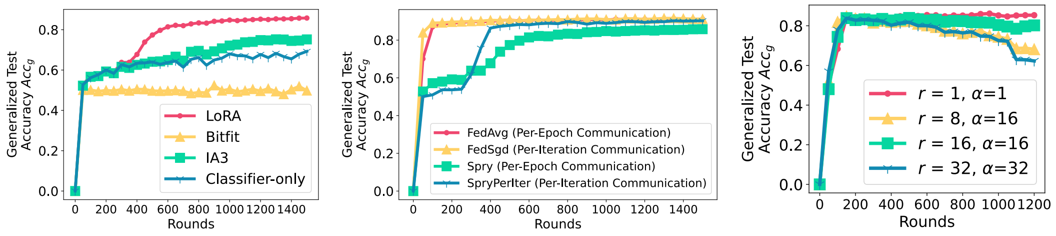
Note that the per-iteration computation cost of SPRY is not the whole picture. It takes fewer communication rounds to reach higher accuracy due to better gradient approximation of forward-mode AD than finite difference methods. This is why "Time to Convergence" (Section 5.3) discusses a fair comparison of SPRY's runtime and prediction performance.

**Server-side per-round computation cost.** On the server side, SPRY is the least computationally demanding. SPRY needs to aggregate a subset of layer weights from only the clients that were assigned to those layers, while its counterparts need to aggregate all layers from all clients.

Computation cost on the server-side changes based on the communication frequency per-iteration communication incurs an additional overhead of $w_\ell L(\frac{M}{L} + 1)$ and $w_\ell L(M + 1)$ (generation of perturbations at the server-side, and multiplying those perturbations with aggregate of the `jvp` values received from the clients) for SPRY and its zero-order counterparts respectively.

Table 4: Generalized ($Acc_g$) and personalized ($Acc_p$) accuracies (the higher, the better) for SPRY and its backpropagation and zero-order based counterparts on various language model architectures. The datasets are split with Dir $\alpha = 0.1$.

| | Backpropgation-based | | | | Zero-order based Method | | First-order Forward-mode AD | |
|---|---|---|---|---|---|---|---|---|
| | FEDAVG | | FEDYOGI | | FWDLLM+ | | SPRY | |
| | $Acc_g$ | $Acc_p$ | $Acc_g$ | $Acc_p$ | $Acc_g$ | $Acc_p$ | $Acc_g$ | $Acc_p$ |
| AG News on BERT Base | 93.00% | 93.34% | 93.31% | 93.88% | 83.41% | 83.42% | 86.74% | 92.42% |
| SST2 on DistilBERT Base | 91.47% | 95.28% | 87.95% | 92.97% | 79.09% | 80.94% | 84.90% | 87.12% |
| SNLI on BERT Large | 85.79% | 89.36% | 86.72% | 90.33% | 66.76% | 64.84% | 77.01% | 77.21% |
| Yahoo on DistilBERT Base | 69.13% | 74.75% | 63.84% | 71.25% | 54.29% | 55.87% | 61.17% | 61.47% |
| Yelp on AlbertV2 Large | 90.17% | 92.78% | 90.24% | 94.00% | 82.65% | 83.25% | 85.80% | 86.00% |



(a) Effect of different PEFT methods on SPRY

(b) Effect of per-epoch and per-iteration communication

(c) Changing LoRA $r$ and $\alpha$ for SPRY

Figure 4: Ablation studies on PEFT methods, communication frequency, and LoRA hyperparameters.

## G Ablation Studies

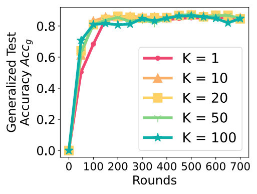
Here we will dive into various components of SPRY and see their impact on the performance of SPRY.

**SPRY can Generalize to Different Language Model Architectures.** Table 4 shows generalized and personalized accuracies $Acc_g$ and $Acc_p$ for homogeneous data splits for language model architectures other than RoBERTa Large.

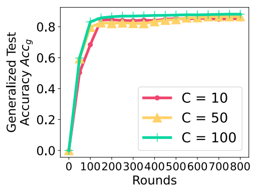
For zero-order gradients, we show the results of the best-performing method, FWDLLM+. We see a similar trend of SPRY outperforming FWDLLM+ by 3.15-10.25% for generalized accuracy, and by 2.75-12.37% for personalized accuracy, exhibiting how SPRY is independent of model architectures. SPRY also comes as close as 4.44-9.71% to the best-performing backpropagation-based method.

**SPRY Supports Other PEFT Methods.** Figure 4a shows the generalized accuracy of SPRY using three different PEFT methods: LoRA, IA3, and BITFIT. We also experiment with finetuning only classifier layers, calling it CLASSIFIER-ONLY. LoRA (with 0.3241% of the total parameters of RoBERTa Large) outperforms IA3 (with 0.3449% of the total parameters) by 10.60%, while BITFIT fails to converge for all datasets. Only training classifier layers is worse than finetuning LoRA weights by 16.53%. The observation on comparison of LoRA with IA3 is consistent with the work benchmarking various PEFT methods [56]. BITFIT has been observed to fail on LLMs [57]. Furthermore, unlike IA3, LoRA has been shown to be successful at finetuning quantized billion-sized models in QLoRA [58], making it a strong candidate for our work.

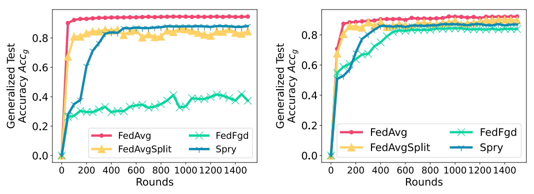
**Effects of Communication Frequency.** One way to reduce the noise introduced by the random perturbations in gradient computation is, to communicate the gradients back to the server every iteration instead of every few epochs. Figure 4b shows results of per-epoch and per-iteration communication variants of SPRY and FEDAVG.



(a) Changing $K$ (Forward-mode AD perturbation count per iteration)



(b) Changing $C$ (Per-round participating client count)



(c) RoBERTa Large (Left) and BERT Base (Right) - Splitting parameters across clients on Backpropagation vs Forward-mode AD

Figure 5: Ablation studies on perturbation counts, participating client counts, and layer splitting strategy.

By communicating each iteration, we see a boost of 4.47% in the accuracy of SPRY, only 0.92% and 0.96% away from the accuracy of FEDAVG and FEDSGD respectively. Furthermore, as shown in Figure 4b, the convergence speed of SPRY also improves by communicating each iteration.

As discussed in Section 3, to reduce the trade-off between performance gains and communication cost per iteration, each client can send only the `jvp` scalar to the server instead of transmitting all the assigned trainable weights [20]. With the seed value of the randomness, the server can then generate the random perturbation vector $\boldsymbol{v}$, which was used by all the clients to generate their respective `jvp` values. Then the random perturbation is multiplied with the received `jvp` values of all clients, to compute gradients.

**Effects of the Number of Trainable Weights.** Figure 4c displays the effect of changing trainable LoRA parameter count on the prediction performance of SPRY. For DistilBERT Base (total parameter count of 66M), LoRA reduces the trainable parameter count to 0.61M (0.91%), 0.74M (1.11%), 0.89M (1.33%), and 1.18M (1.77%) with LoRA hyperparameter settings of ($r$=1, $\alpha$=1), ($r$=8, $\alpha$=16), ($r$=16, $\alpha$=16), and ($r$=32, $\alpha$=32).

SPRY achieves the highest accuracy of 84.90% with ($r$=1, $\alpha$=1) setting, which has the smallest trainable parameter count. The accuracy increases as the layer size decreases since fewer perturbed weights provide less noisy gradients.

**Effects of the Number of Perturbations per Batch.** The effect of increasing the number of perturbations per batch and hence the number of `jvp` evaluations for a batch is shown in Figure 5a for the SST2 dataset. Here, gradients generated from each random perturbation and their corresponding `jvp` values are averaged to update the model. We observe that increasing $K$ (perturbations per batch) for Forward-mode AD has little to no impact on the end prediction performance of SPRY, with $K = 100$ improving the generalized test accuracy by 1.1% over $K = 1$. However, the benefits of increasing the perturbation count per batch are seen in the convergence speed. Setting $K = 10$ achieves a steady state (of accuracy $\sim$86%) around 200th round, while the setting with $K = 1$ takes 500 rounds. The improvements in convergence speed are saturated for $K > 10$. This shows that more perturbations reduce the gradient estimation noise only to an extent.

**Effects of the Number of Participating Clients per Round.** Figure 5b shows how changing the per-round number of participating clients $C$ influences SPRY on the SST2 dataset. Increasing client count increases the prediction performance of SPRY. With the total client count fixed to 100, the three settings $C = 10$, $C = 50$, and $C = 100$ produce accuracies of 85.14%, 86.56%, and 88.08%, respectively. Similar to the findings of Section G, we also see an improvement in the convergence speed as the participating client count increases. To achieve an accuracy of $\sim$85%; $C = 10$, $C = 50$, $C = 100$ require 500, 450, and 150 rounds respectively. The performance gains and faster convergence are due to more clients training the weights of the same layers.

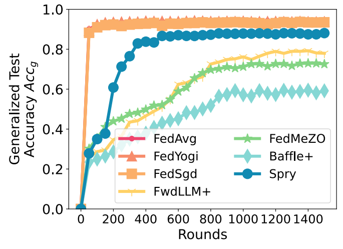
**The Importance of Splitting Layers.** To understand the effects of splitting, we compare the results of the following two experiments: (a) With FEDAVGSPLIT, we apply the strategy of splitting trainable layers across clients (Section 3.1) to backpropagation-based FEDAVG, and (b) With FEDFGD, we omit the splitting strategy of SPRY.

Figure 5c shows the performance of FEDAVG and FEDAVGSPLIT against FEDFGD and SPRY for two LMs: RoBERTa Large (355M) and BERT Base (110M). We observe that FEDAVGSPLIT fails to achieve similar accuracy for both models with a drop of 2.60% and 10.00%. This is because in FEDAVGSPLIT, fewer clients are training each subset of weights. Moreover, we see a similar accuracy with an absolute difference of 2.70-3.61% between FEDAVGSPLIT and SPRY, since the trainable weight count per client is low. FEDYOGISPLIT follows the same observation of not achieving similar accuracy to FEDYOGI if the trainable weights are split across clients. On the contrary, FEDFGD converges for the smaller model BERT base, albeit 150 rounds slower than SPRY, and with 2.87% accuracy drop. But as the size of trainable weights increases, e.g., for RoBERTa Large, FEDFGD fails to converge. This proves the necessity of splitting layers for Forward-mode AD so that each client has fewer trainable weights to perturb.

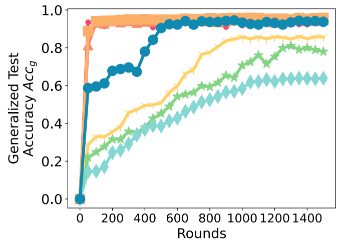
## H Additional Results

### H.1 Generalized Performance Curves

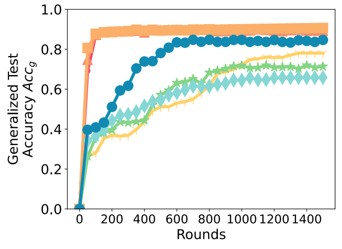
Generalized results on homogeneous and heterogeneous clients with Dir $\alpha = 1.0$ and $\alpha = 0.1$ are shown in (a) Figures 6 and 8 for RoBERTa Large, Llama2-7B, OPT6.7B, OPT13B; and (b) Figure 7 for BERT Large, BERT Base, DistilBert Base, Albert Large v2.



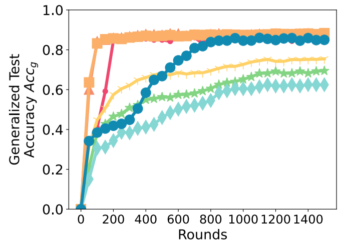
(a) AG News with RoBERTa Large



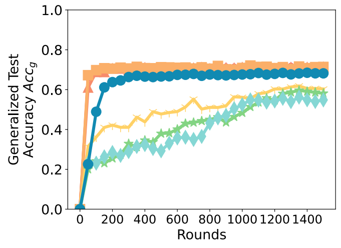
(b) SST2 with RoBERTa Large



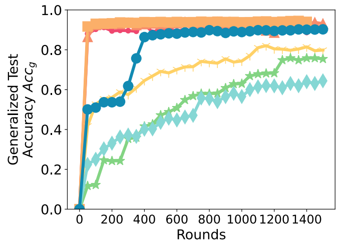
(c) SNLI with RoBERTa Large



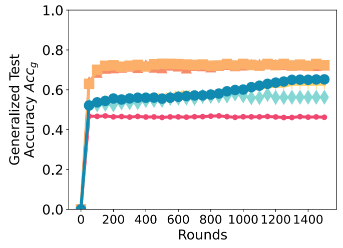
(d) MNLI with RoBERTa Large



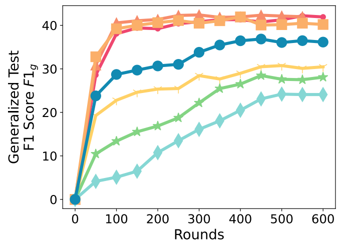
(e) Yahoo with RoBERTa Large



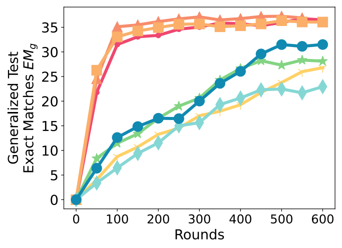
(f) Yelp with RoBERTa Large



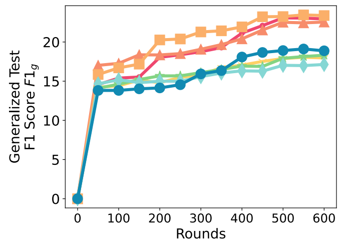
(g) MultiRC with Llama2-7B



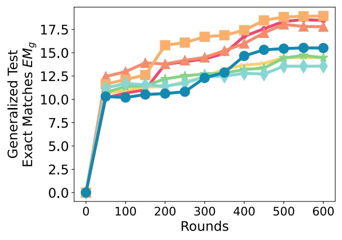
(h) SQuADv2 with OPT6.7B



(i) SQuADv2 with OPT6.7B

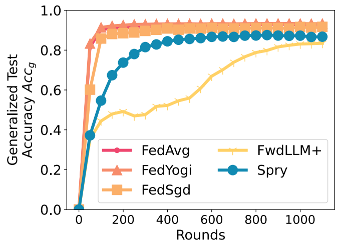


(j) SQuADv2 with OPT13B

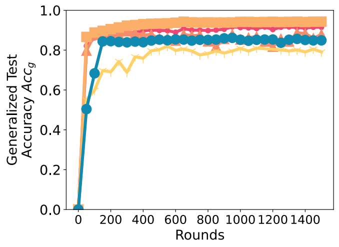


(k) SQuADv2 with OPT13B

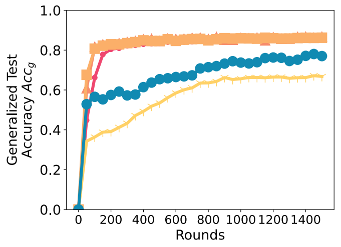
Figure 6: Generalized accuracy / F1 score / Exact matches for homogeneous clients (Dirichlet $\alpha = 1.0$) setting



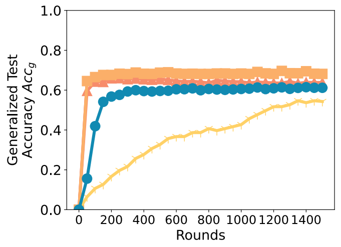
(a) AG News with BERT Base



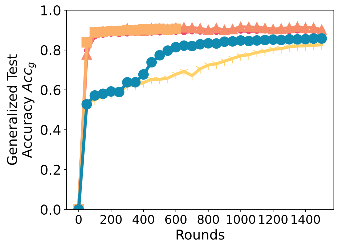
(b) SST2 with DistilBERT Base



(c) SNLI with BERT Large

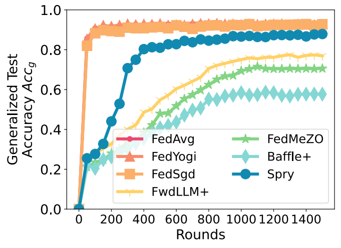


(d) Yahoo with DistilBERT Base

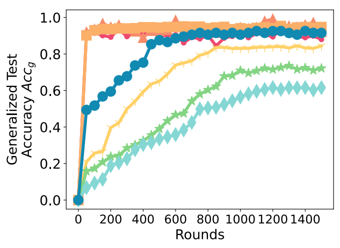


(e) Yelp with Albert Large v2

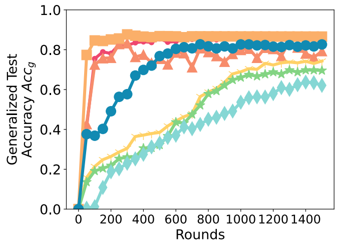
Figure 7: Generalized accuracy /F1 score / Exact matches for
homogeneous clients (Dirichlet $\alpha = 1.0$) setting for a variety of language models



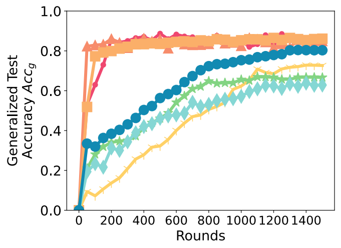
(a) AG News with RoBERTa Large



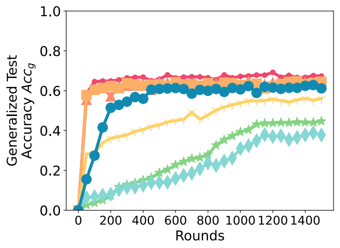
(b) SST2 with RoBERTa Large



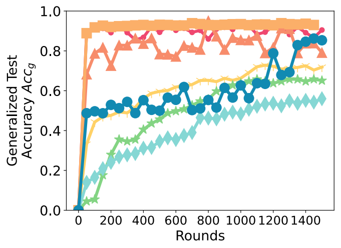
(c) SNLI with RoBERTa Large



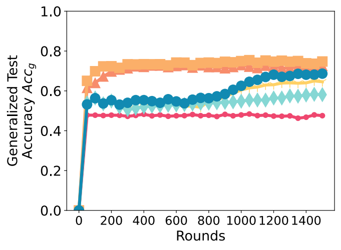
(d) MNLI with RoBERTa Large



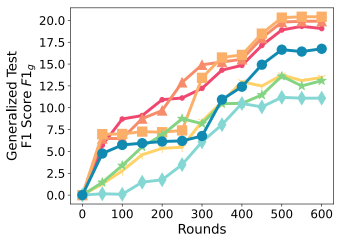
(e) Yahoo with RoBERTa Large



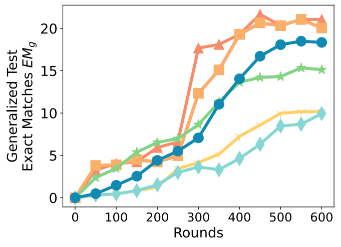
(f) Yelp with RoBERTa Large



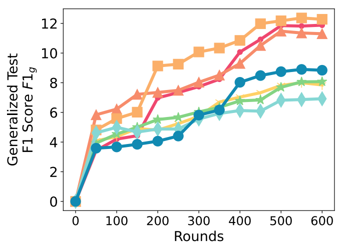
(g) MultiRC with Llama2-7B



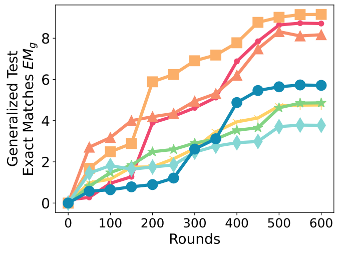
(h) SQuADv2 with OPT6.7B



(i) SQuADv2 with OPT6.7B



(j) SQuADv2 with OPT13B



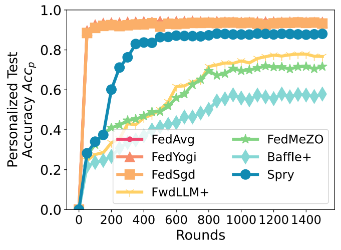
(k) SQuADv2 with OPT13B

Figure 8: Generalized accuracy / F1 score / Exact matches for heterogeneous clients (Dirichlet $\alpha = 0.1$) setting

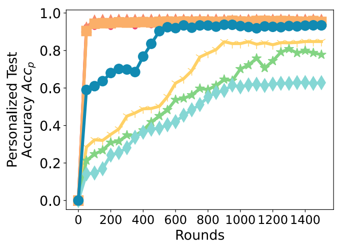
## H.2 Personalized Performance Curves

Personalized results on homogeneous and heterogeneous clients with Dir $\alpha = 1.0$ and $\alpha = 0.1$ are shown in (a) Figures 9 and 11 for RoBERTa Large, Llama2-7B, OPT6.7B, OPT13B; and (b) Figure 10 for BERT Large, BERT Base, DistilBert Base, Albert Large v2.

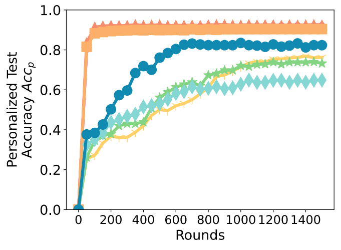
Table 5 shows accuracy and F1 scores of SPRY and its backpropagation and zero-order based counterparts.



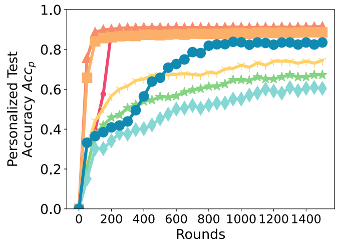
(a) AG News with RoBERTa Large



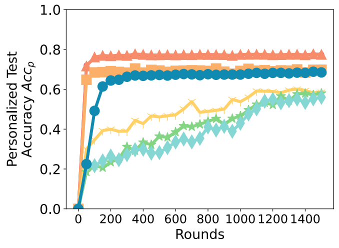
(b) SST2 with RoBERTa Large



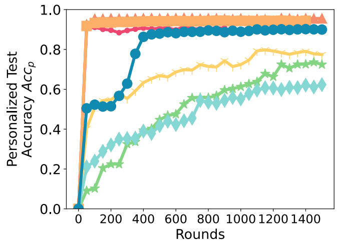
(c) SNLI with RoBERTa Large



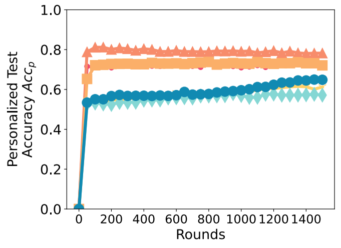
(d) MNLI with RoBERTa Large



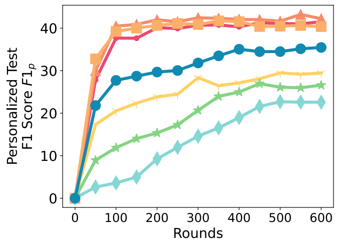
(e) Yahoo with RoBERTa Large



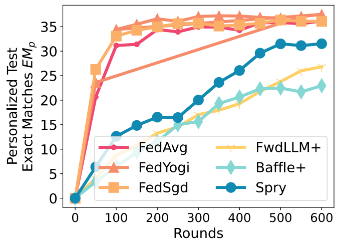
(f) Yelp with RoBERTa Large



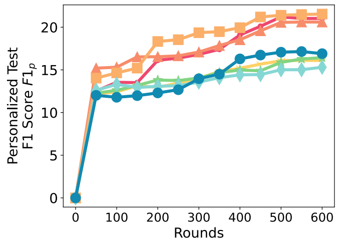
(g) MultiRC with Llama2-7B



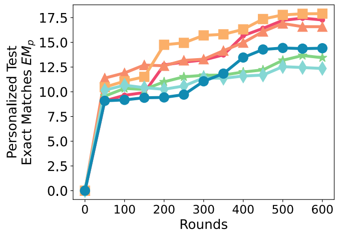
(h) SQuADv2 with OPT6.7B



(i) SQuADv2 with OPT6.7B

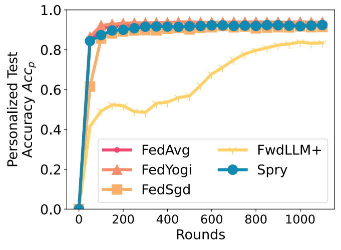


(j) SQuADv2 with OPT13B

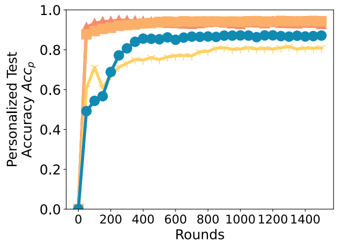


(k) SQuADv2 with OPT13B

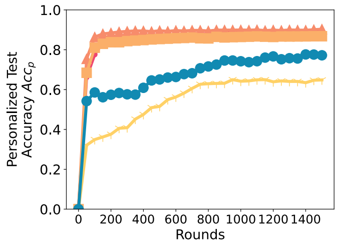
Figure 9: Personalized accuracy / F1 score / Exact matches for homogeneous clients (Dirichlet $\alpha = 1.0$) setting



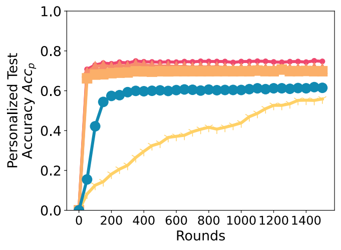
(a) AG News with BERT Base



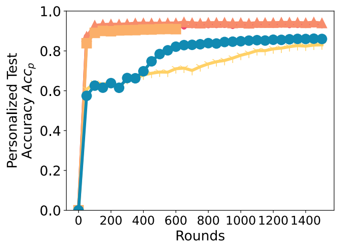
(b) SST2 with DistilBERT Base



(c) SNLI with BERT Large



(d) Yahoo with DistilBERT Base

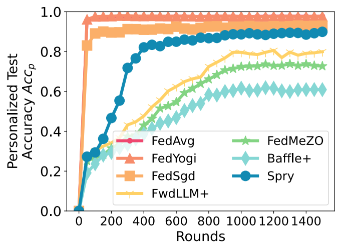


(e) Yelp with Albert Large v2

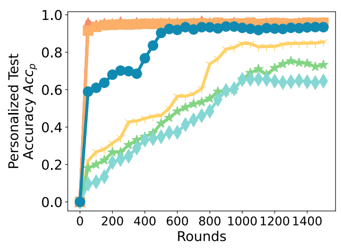
Figure 10: Personalized accuracy / F1 score / Exact matches for homogeneous clients (Dirichlet $\alpha = 1.0$) setting for a variety of language models

Table 5: Personalized accuracy ($Acc_p$) for SPRY and its backpropagation- and zero-order-based counterparts on RoBERTa Large and LLMs. SQuADv2 uses F1 score. ↑ shows that higher values are better. The datasets are split with Dir $\alpha = 0.1$. $\diamond$ = Llama2-7B. $\star$ = OPT6.7B. □ = OPT13B. SPRY significantly outperforms the best-performing zero-order-based methods.

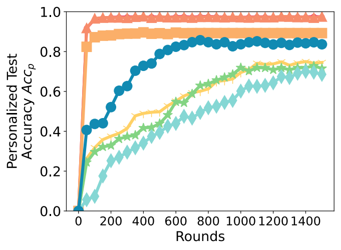
| | Backpropagation-based Methods ↑ | | Zero-order-based Methods ↑ | | | First-order Forward Mode AD ↑ | Difference between performances of SPRY and | |
|---|---|---|---|---|---|---|---|---|
| | FEDAVG | FEDYOGI | FWDLLM+ | FEDMEZO | BAFFLE+ | SPRY | best-performing backpropagation method ↑ | best-performing zero-order method ↑ |
| AG News | 97.76% | 97.71% | 79.94% | 72.69% | 60.89% | 89.91% | -7.85% | 9.97% |
| SST2 | 95.84% | 95.90% | 85.51% | 73.26% | 64.55% | 93.40% | -2.50% | 7.89% |
| SNLI | 97.41% | 97.57% | 74.53% | 71.54% | 68.55% | 83.45% | -14.12% | 8.92% |
| MNLI | 90.38% | 90.03% | 72.71% | 67.53% | 63.58% | 80.63% | -9.75% | 7.92% |
| Yahoo | 89.76% | 89.64% | 77.93% | 67.64% | 59.40% | 82.80% | -6.96% | 4.87% |
| Yelp | 93.44% | 97.81% | 73.04% | 68.77% | 57.78% | 85.83% | -11.98% | 12.79% |
| MultiRC $\diamond$ | 50.28% | 75.21% | 65.91% | N/A | 61.41% | 71.20% | -4.01% | 5.29% |
| SQuADv2 $\star$ | 20.49 | 21.25 | 14.43 | 14.04 | 12.27 | 17.73 | -3.52 | 3.30 |
| SQuADv2 □ | 13.06 | 12.49 | 8.96 | 9.10 | 8.17 | 9.88 | -3.18 | 0.78 |



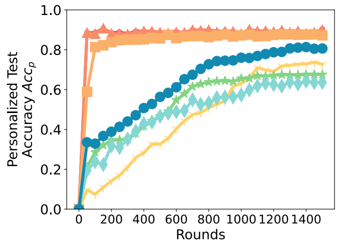
(a) AG News with RoBERTa Large



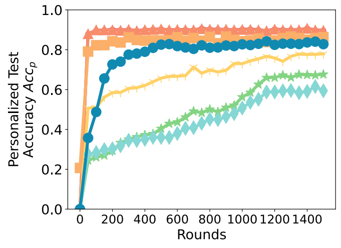
(b) SST2 with RoBERTa Large



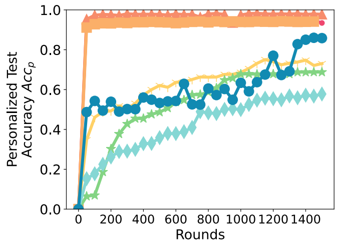
(c) SNLI with RoBERTa Large



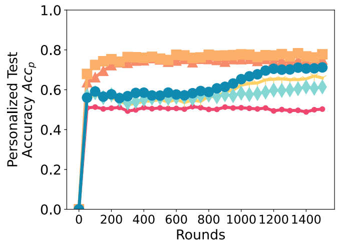
(d) MNLI with RoBERTa Large



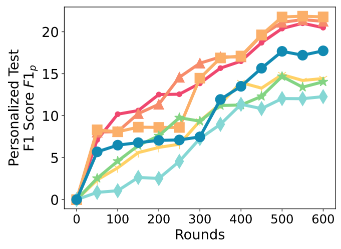
(e) Yahoo with RoBERTa Large



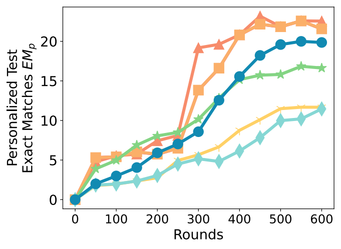
(f) Yelp with RoBERTa Large



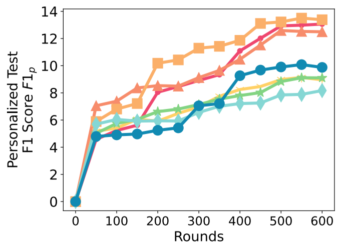
(g) MultiRC with Llama2-7B



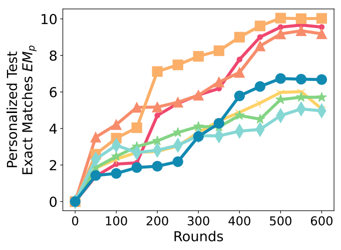
(h) SQuADv2 with OPT6.7B



(i) SQuADv2 with OPT6.7B



(j) SQuADv2 with OPT13B



(k) SQuADv2 with OPT13B

Figure 11: Personalized accuracy / F1 score / Exact matches for heterogeneous clients (Dirichlet $\alpha = 0.1$) setting

## H.3 Experiment Variance

Tables 6 and 7 show the variance of running the same experiments thrice with the random seeds 0, 1, and 2.

Table 6: Experimental variance (±) for SPRY and its counterparts on RoBERTa Large.

| | Backpropgation-based Methods | | | | Zero-order-based Methods | | | | | | First-order Forward-mode AD | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | FEDAVG | | FEDYOGI | | FWDLLM+ | | FEDMEZO | | BAFFLE+ | | SPRY | |
| | $Acc_g$ | $Acc_p$ | $Acc_g$ | $Acc_p$ | $Acc_g$ | $Acc_p$ | $Acc_g$ | $Acc_p$ | $Acc_g$ | $Acc_p$ | $Acc_g$ | $Acc_p$ |
| AG News | 0.51% | 0.44% | 0.26% | 0.21% | 0.73% | 0.71% | 1.34% | 1.27% | 0.68% | 0.51% | 1.16% | 1.08% |
| SST2 | 0.37% | 0.47% | 0.43% | 0.37% | 1.26% | 1.14% | 1.13% | 1.06% | 0.41% | 0.45% | 0.77% | 0.95% |
| SNLI | 0.45% | 0.39% | 0.17% | 0.11% | 0.82% | 0.67% | 0.74% | 0.65% | 0.84% | 0.78% | 0.51% | 0.45% |
| MNLI | 0.63% | 0.58% | 0.29% | 0.24% | 1.39% | 1.25% | 1.98% | 1.85% | 1.15% | 1.01% | 1.45% | 1.32% |
| Yahoo | 0.24% | 0.33% | 0.53% | 0.47% | 0.47% | 0.44% | 1.06% | 0.93% | 0.59% | 0.48% | 0.99% | 0.85% |
| Yelp | 0.22% | 0.25% | 0.36% | 0.27% | 0.54% | 0.49% | 0.82% | 0.66% | 0.83% | 0.71% | 0.76% | 0.61% |

Table 7: Experimental variance (±) for generalized ($Acc_g$ for MultiRC / $F1_g$ for SQuADv2) and personalized ($Acc_p$ for MultiRC / $F1_p$ for SQuADv2) accuracy or F1 score for SPRY and its counterparts. $\diamond$ = Llama2 7B. $\star$ = OPT 6.7B. $\square$ = OPT 13B.

| | Backpropgation-based Methods | | | | Zero-order-based Methods | | | | | | First-order Forward-mode AD | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | FEDAVG | | FEDYOGI | | FWDLLM+ | | FEDMEZO | | BAFFLE+ | | SPRY | |
| | $Acc_g$ | $Acc_p$ | $Acc_g$ | $Acc_p$ | $Acc_g$ | $Acc_p$ | $Acc_g$ | $Acc_p$ | $Acc_g$ | $Acc_p$ | $Acc_g$ | $Acc_p$ |
| MultiRC $\diamond$ | 0.58% | 0.43% | 0.34% | 0.29% | 0.89% | 0.74% | N/A | N/A | 1.34% | 1.02% | 0.97% | 0.81% |
| SQuADv2 $\star$ | 1.73 | 1.07 | 1.42 | 0.84 | 2.17 | 1.87 | 1.78 | 1.51 | 2.78 | 2.01 | 1.76 | 0.99 |
| SQuADv2 $\square$ | 0.86 | 0.43 | 0.61 | 0.45 | 1.16 | 0.97 | 1.03 | 0.87 | 1.34 | 1.14 | 0.97 | 0.81 |

# I Proofs

## I.1 Basics

**Server Update.** The server update of SPRY uses adaptive optimizer FEDYOGI. However, to simplify the proofs without losing generality, we use the server update of FEDADAM [25]. FEDADAM has the exact update rule as FEDYOGI but without a `sign` function in its calculation of the second moment of the gradients (See Algorithm 2 of AFO [25]).

Hence, the server update of SPRY is

$$w^{(r)} \leftarrow w^{(r-1)} + \eta \frac{\Delta^{(r)}}{\sqrt{\Lambda^{(r)}} + \tau} \qquad \forall \text{ trainable weights } w^{(r)} \in [d]. \tag{7}$$

$\Delta^{(r)}$ is the square of accumulated gradients from all clients. $\Lambda^{(r)}$ is the second moment of $\Delta^{(r)}$. $\tau$ is a small positive real number, to prevent division by zero errors. Note that we are assuming flattened weights $w \in \mathbb{R}^d$ without the loss of generality.

With SPRY, the aim is to solve the following optimization problem:

$$\min_{w \in \mathbb{R}^d} f(w) = \frac{1}{m} \sum_{m=1}^{M} F_m(w_{\overline{m}}), \tag{8}$$

where $\overline{m} \in \left[\frac{(m-1)d}{M} + 1, \frac{md}{M}\right]$, $F_m(w_{\overline{m}}) = \mathbb{E}_{(x,y)\sim\mathcal{D}_m}[f_m(w_{\overline{m}}, (x, y))]$ is the objective function, $\mathcal{D}_m$ is the dataset, and $f_m$ is the loss function of a client $m \in [M]$.

**Accumulated Gradients.** With SPRY, each client trains a subset of weights $w_{\overline{m}}$. In a low participation rate setting, each $w_{\overline{m}}$ is only trained by one of the participating clients from the set of available clients $\mathcal{M}$. Although we make our analysis more generally applicable by showing multiple clients training the same $w_{\overline{m}}$.

The true global gradients can be written as,

$$\nabla f(w) = \left[\frac{1}{\widetilde{M}} \sum_{m \in \widetilde{\mathcal{M}}} \nabla F(w_{\overline{m}}) \,\Big|\, \overline{m} \in \left[\frac{(m-1)d}{M} + 1, \frac{md}{M}\right], \widetilde{\mathcal{M}} \subset \mathcal{M}\right] \tag{9}$$

where $\widetilde{\mathcal{M}}$ is a set of clients training the subset of the weights $w_{\overline{m}}$. $\widetilde{M}$ is the size of $\widetilde{\mathcal{M}}$.

**Client Update.** The directional derivative of Forward-mode AD is denoted as $\nabla \hat{f}_v(w; (x, y))$, where $v \in \mathbb{R}^d$ is the random perturbation of weights $w$ and $(x, y)$ are sampled from a dataset $\mathcal{D}$. For each client $m$, through Forward-mode AD, we have $\nabla \hat{f}_m(w_{\overline{m}}, v_{\overline{m}}; (x, y)) = (\nabla \hat{f}_{m,v}(w_{\overline{m}}; (x, y)) \cdot v_{\overline{m}})$ to estimate the true gradient $\nabla F_m(w_{\overline{m}})$:

$$\mathbb{E}_{v_{\overline{m}},\mathcal{D}_m}\left[\nabla \hat{f}_m(w_{\overline{m}}, v_{\overline{m}}; \mathcal{D}_m)\right] = \frac{1}{DK} \sum_{(x,y)\sim\mathcal{D}_m} \sum_{i=1}^{K} \mathbb{E}_{v_{i,\overline{m}},(x,y)}\left[\nabla f_m(w_{\overline{m}}; (x, y)) v_{i,\overline{m}} v_{i,\overline{m}}^T\right] \tag{10}$$

$$\mathbb{E}_{v_{\overline{m}},\mathcal{D}_m}\left[\nabla \hat{f}_m(w_{\overline{m}}, v_{\overline{m}}; \mathcal{D}_m)\right] = \nabla F_m(w_{\overline{m}}) \qquad \because \text{Theorem 1 of FGD [23]} \tag{11}$$

Here, the expectation is under the randomness of sampled data $\mathcal{D}$, and random perturbation $v$. $K$ is the number of perturbations per batch. SPRY uses $K = 1$ by default, but here we aim to make our analysis more general to see the impact of $K$ on various properties of SPRY.

## I.2 Assumptions

We will be using the following assumptions to derive the first and second moment of the forward gradient and to calculate the rate of convergence of SPRY,

**Assumption I.1** (Smoothness)**.** The gradient of function $F_m$ is $L$-Lipschitz,

$$||\nabla F_m(w_1) - \nabla F_m(w_2)|| \leq L||w_1 - w_2||;\ \forall m \in [M] \text{ and } w_1, w_2 \in \mathbb{R}^d.$$

**Assumption I.2** (Bounded Global Variance (Assumption 2 in AFO [25])**.** The variance of the global objective function $f$ is bounded by $\sigma_g$ as

$$\frac{1}{M}\sum_{m=1}^{M} || \left[\nabla F_m(w_{\overline{m}})\right]_j - \left[\nabla f(w_{\overline{m}})\right]_j ||^2 \le \sigma_{g,j}^2;\ \forall m \in [M], w \in \mathbb{R}^d \text{ and } \forall j \in [d],$$

where $\nabla F_m$ is the true gradient of client $m$. As defined earlier, $\overline{m} \in \left[\frac{(m-1)d}{M} + 1, \frac{md}{M}\right]$.

**Assumption I.3** (Bounded Gradients (Assumption 3 in AFO [25])**.** The function $f_m(w;(x,y))$ has $G$-bounded gradients such that for any client $m \in [M]$, weights $w \in \mathbb{R}^d$, and sampled data $(x,y) \sim \mathcal{D}_m$; we have

$$|\left[\nabla f_m(w;(x,y))\right]_j| < G,\ \forall j \in [d]$$

## I.3 First and Second Moments of Forward Gradients

We first prove that in SPRY, estimation of $\nabla f$ by the accumulated forward mode gradients $\nabla \hat{f}$ across all clients of an arbitrary round $r$, depends on the heterogeneity across client datasets. Statements on homogeneous data have been proven for FGD [23] and MEZO [18] in single-client or centralized settings. Here we focus on the specific federated setting of SPRY, where gradients are accumulated differently than in traditional FEDAVG [1]. In SPRY, we have

$$\nabla f(w) = \mathbb{E}_v\left[\nabla \hat{f}(w,v)\right] = \left[\frac{1}{\widetilde{M}}\sum_{m\in\widetilde{\mathcal{M}}} \mathbb{E}\left[\nabla \hat{f}_m(w_{\overline{m}}, v_{\overline{m}}; \mathcal{D}_m)\right] \middle|\ \forall \widetilde{\mathcal{M}} \subset \mathcal{M}; \overline{m} \in \left[\frac{(m-1)d}{M}+1, \frac{md}{M}\right]\right], \tag{12}$$

where $w$ represents weights, $v$ are their corresponding perturbations, and $\mathcal{D}_m$ is the dataset of client $m$.

We omit the round index of the model weights $w$ and their perturbations $v$ for this section since the same relationship will hold for any arbitrary round $r$.

**Theorem I.4** (Estimation of the Global Gradient)**.** *In* SPRY*, global forward gradient $\nabla \hat{f}$ of the trainable weights $w \in \mathbb{R}^d$, with the corresponding weight perturbations $v \in \mathbb{R}^d$, computed by $M$ participating clients is estimated in terms of true global gradient $\nabla f$ as,*

$$\begin{aligned}\mathbb{E}_{v,\mathcal{D}}[\nabla \hat{f}(w,v;\mathcal{D})] &= \nabla f(w) \\ &+ \frac{1}{\widetilde{M}}\begin{bmatrix} \sum_{m\in\widetilde{\mathcal{M}}_1}\sum_{c=1}^{C}\alpha_{m,c}\mathbb{E}_{(x,y_c)\in\mathcal{D}}\left[\nabla \hat{f}_m(w_{[1,\frac{d}{M}]}, v_{[1,\frac{d}{M}]}; (x,y_c))\right] \\ \sum_{m\in\widetilde{\mathcal{M}}_2}\sum_{c=1}^{C}\alpha_{m,c}\mathbb{E}_{(x,y_c)\in\mathcal{D}}\left[\nabla \hat{f}_m(w_{[\frac{d}{M}+1,\frac{2d}{M}]}, v_{[\frac{d}{M}+1,\frac{2d}{M}]}; (x,y_c))\right] \\ \vdots \end{bmatrix}^T\end{aligned}$$

*where $C$ is total number of classes and $\alpha_{m,c} = \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)$. For a class $c$, $n_c$ is its sample count, $\alpha_c$ is its Dirichlet concentration parameter. For a client $m$; $n_{m,c}$ is the sample count of the $c^{th}$ class, and $\mathcal{D}_m$ is the size of the data of client $m$. The global data is $\mathcal{D} = \sum_{m\in\mathcal{M}}\mathcal{D}_m$. $\widetilde{\mathcal{M}}$ is the set of clients training an arbitrary subset of weights, $\widetilde{M} = |\widetilde{\mathcal{M}}_i|;\ \forall i \in [M/d]$.*

*Proof.* Suppose the global dataset $\mathcal{D}$ is defined as a combination of all $\mathcal{D}_m$. Hence, $\mathcal{D} = \cup_{m\in[M]}\mathcal{D}_m$.

In SPRY, the global forward gradient is defined as

$$\mathbb{E}_{v,\mathcal{D}}[\nabla \hat{f}(w,v;\mathcal{D})] = \mathbb{E}\begin{bmatrix} \frac{1}{\widetilde{M}}\sum_{m\in\widetilde{\mathcal{M}}_1}\nabla \hat{f}_m(w_{[1,\frac{d}{M}]}, v_{[1,\frac{d}{M}]}; \mathcal{D}) \\ \frac{1}{\widetilde{M}}\sum_{m\in\widetilde{\mathcal{M}}_2}\nabla \hat{f}_m(w_{[\frac{d}{M}+1,\frac{2d}{M}]}, v_{[\frac{d}{M}+1,\frac{2d}{M}]}; \mathcal{D}) \\ \vdots \\ \frac{1}{\widetilde{M}}\sum_{m\in\widetilde{\mathcal{M}}_{M/d}}\nabla \hat{f}_m(w_{[\frac{(M-1)d}{M}+1,d]}, v_{[\frac{(M-1)d}{M}+1,d]}; \mathcal{D}) \end{bmatrix}^T \tag{13}$$

To combine the gradient estimates from the above equation for all clients $m \in \mathcal{M}$, we first consider the *bias* $b_m$ between the gradient estimate under **(a)** Globally combined data $\mathcal{D}$ and **(b)** Data $\mathcal{D}_m$ of an arbitrary client $m$.

**Measuring the dataset bias.** Note that we consider samples $(x, y)$ of $\mathcal{D}_m$ to be sampled from $\mathcal{D}$, since $\mathcal{D}_m \subset \mathcal{D}$.

$$\mathbb{E}_{v_{\overline{m}},\mathcal{D}}\left[\nabla \hat{f}_m(w_{\overline{m}}, v_{\overline{m}}; \mathcal{D})\right] = \mathbb{E}_{v_{\overline{m}},\mathcal{D}_m}\left[\nabla \hat{f}_m(w_{\overline{m}}, v_{\overline{m}}; \mathcal{D}_m)\right] + b_m \tag{14}$$

Computing the *bias* term $b$ requires information on dataset distribution. In FL settings, Dirichlet distribution to spread the data into heterogeneous splits is a popular way to simulate heterogeneity [37]. The biased dataset $\mathcal{D}_m$ is sampled from the combined dataset $\mathcal{D}$ using the Dirichlet distribution. This allows us to utilize the properties of the distribution to derive the relationship between forward gradients on global data and on individual local data:

For classification tasks, let's say $\mathcal{D}$ has $C$ total classes: $y_1, \ldots, y_C$. We have $\alpha_1, \ldots, \alpha_C$ as concentration parameters of the Dirichlet distribution. The expected forward gradient for the combined dataset $\mathcal{D}$ can be expressed as a weighted sum of the expected forward gradients for each class $C$:

$$\mathbb{E}_{v_{\overline{m}},\mathcal{D}}\left[\nabla \hat{f}_m(w_{\overline{m}}, v_{\overline{m}}; \mathcal{D})\right] = \sum_{c=1}^{C} \frac{n_c}{|\mathcal{D}|} \mathbb{E}_{v_{\overline{m}},(x,y_c)\in\mathcal{D}}\left[\nabla \hat{f}_m(w_{\overline{m}}, v_{\overline{m}}; (x, y_c))\right], \tag{15}$$

where $n_c$ is the sample count of class $c$. And for the biased dataset $\mathcal{D}_m$ sampled with Dirichlet concentration parameters $\alpha_1, \ldots, \alpha_C$, the expected forward gradient can be expressed as,

$$\mathbb{E}_{v_{\overline{m}},\mathcal{D}_m}\left[\nabla \hat{f}_m(w_{\overline{m}}, v_{\overline{m}}; \mathcal{D}_m)\right] = \sum_{c=1}^{C} \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|} \mathbb{E}_{v_{\overline{m}},(x,y_c)\in\mathcal{D}}\left[\nabla \hat{f}_m(w_{\overline{m}}, v_{\overline{m}}; (x, y_c))\right], \tag{16}$$

where $n_{m,c}$ is the sample count of class $c$ for client $m$.

Subtracting Equation 16 from Equation 15,

$$b_m = \mathbb{E}_{v_{\overline{m}},\mathcal{D}}\left[\nabla \hat{f}_m(w_{\overline{m}}, v_{\overline{m}}; \mathcal{D})\right] - \mathbb{E}_{v_{\overline{m}},\mathcal{D}_m}\left[\nabla \hat{f}_m(w_{\overline{m}}, v_{\overline{m}}; \mathcal{D}_m)\right] \tag{17}$$

$$= \sum_{c=1}^{C} \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right) \mathbb{E}_{v_{\overline{m}},(x,y_c)\in\mathcal{D}}\left[\nabla \hat{f}_m(w_{\overline{m}}, v_{\overline{m}}; (x, y_c))\right] \tag{18}$$

For any $\overline{m} \in \left[\frac{(m-1)d}{M} + 1, \frac{md}{M}\right]$, we have $\mathbb{E}_{v_{\overline{m}},\mathcal{D}_m}\left[\nabla \hat{f}_m(w_{\overline{m}}, v_{\overline{m}}; \mathcal{D}_m)\right] = F_m(w)$ from Lemma 1 in FGD [23].

Plugging in the above result and Equation 18 in Equation 14,

$$\mathbb{E}_{v_{\overline{m}},\mathcal{D}}\left[\nabla \hat{f}_m(w_{\overline{m}}, v_{\overline{m}}; \mathcal{D})\right] = \mathbb{E}_{v_{\overline{m}},\mathcal{D}_m}\left[\nabla \hat{f}_m(w_{\overline{m}}, v_{\overline{m}}; \mathcal{D}_m)\right] + b_m$$

$$= \nabla F_m(w_{\overline{m}}) + \sum_{c=1}^{C} \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right) \mathbb{E}_{v_{\overline{m}},(x,y_c)\in\mathcal{D}}\left[\nabla \hat{f}_m(w_{\overline{m}}, v_{\overline{m}}; (x, y_c))\right] \tag{19}$$

For $i^{th}$ row in Equation 13,

$$\mathbb{E}_{v_{\overline{m}},\mathcal{D}}\left[\nabla \hat{f}_i(w_{\overline{m}}, v_{\overline{m}}; \mathcal{D})\right] = \frac{1}{\widetilde{M}} \sum_{m\in\widetilde{\mathcal{M}}_1} \mathbb{E}_{v_{\overline{m}},\mathcal{D}}\left[\nabla \hat{f}_m(w_{\overline{m}}, v_{\overline{m}}; \mathcal{D})\right] \tag{20}$$

$$= \frac{1}{\widetilde{M}} \sum_{m\in\widetilde{\mathcal{M}}_1} \left(\nabla F_m(w_{\overline{m}}) + \sum_{c=1}^{C} \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right) \mathbb{E}_{v_{\overline{m}},(x,y_c)\in\mathcal{D}}\left[\nabla \hat{f}_m(w_{\overline{m}}, v_{\overline{m}}; (x, y_c))\right]\right) \tag{21}$$

$$= \nabla f(w_{\overline{m}}) + \frac{1}{\widetilde{M}} \sum_{m\in\widetilde{\mathcal{M}}_1} \sum_{c=1}^{C} \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right) \mathbb{E}_{v_{\overline{m}},(x,y_c)\in\mathcal{D}}\left[\nabla \hat{f}_m(w_{\overline{m}}, v_{\overline{m}}; (x, y_c))\right] \tag{22}$$

Putting Equation 22 in Equation 13,

$$\mathbb{E}_{v,\mathcal{D}}[\nabla \hat{f}(w,v;\mathcal{D})] \tag{23}$$

$$= \begin{bmatrix} \nabla f(w_{[1,\frac{d}{M}]}) + \frac{1}{\widetilde{M}} \sum_{m\in\widetilde{\mathcal{M}}_1} \sum_{c=1}^{C} \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right) \mathbb{E}_{(x,y_c)\in\mathcal{D}} \left[\nabla \hat{f}_m(w_{[1,\frac{d}{M}]}, v_{[1,\frac{d}{M}]}; (x,y_c))\right] \\ \nabla f(w_{[\frac{d}{M}+1,\frac{2d}{M}]}) + \frac{1}{\widetilde{M}} \sum_{m\in\widetilde{\mathcal{M}}_2} \sum_{c=1}^{C} \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right) \mathbb{E}_{(x,y_c)\in\mathcal{D}} \left[\nabla \hat{f}_m(w_{[\frac{d}{M}+1,\frac{2d}{M}]}, v_{[\frac{d}{M}+1,\frac{2d}{M}]}; (x,y_c))\right] \\ \vdots \end{bmatrix}^T \tag{24}$$

$$= \begin{bmatrix} \nabla f(w_{[1,\frac{d}{M}]}) \\ \nabla f(w_{[\frac{d}{M}+1,\frac{2d}{M}]}) \\ \vdots \end{bmatrix}^T + \frac{1}{\widetilde{M}} \begin{bmatrix} \sum_{m\in\widetilde{\mathcal{M}}_1} \sum_{c=1}^{C} \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right) \mathbb{E}_{(x,y_c)\in\mathcal{D}} \left[\nabla \hat{f}_m(w_{[1,\frac{d}{M}]}, v_{[1,\frac{d}{M}]}; (x,y_c))\right] \\ \sum_{m\in\widetilde{\mathcal{M}}_2} \sum_{c=1}^{C} \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right) \mathbb{E}_{(x,y_c)\in\mathcal{D}} \left[\nabla \hat{f}_m(w_{[\frac{d}{M}+1,\frac{2d}{M}]}, v_{[\frac{d}{M}+1,\frac{2d}{M}]}; (x,y_c))\right] \\ \vdots \end{bmatrix}^T \tag{25}$$

$$= \nabla f(w) + \frac{1}{\widetilde{M}} \begin{bmatrix} \sum_{m\in\widetilde{\mathcal{M}}_1} \sum_{c=1}^{C} \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right) \mathbb{E}_{(x,y_c)\in\mathcal{D}} \left[\nabla \hat{f}_m(w_{[1,\frac{d}{M}]}, v_{[1,\frac{d}{M}]}; (x,y_c))\right] \\ \sum_{m\in\widetilde{\mathcal{M}}_2} \sum_{c=1}^{C} \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right) \mathbb{E}_{(x,y_c)\in\mathcal{D}} \left[\nabla \hat{f}_m(w_{[\frac{d}{M}+1,\frac{2d}{M}]}, v_{[\frac{d}{M}+1,\frac{2d}{M}]}; (x,y_c))\right] \\ \vdots \end{bmatrix}^T \tag{26}$$

□

Next, we formulate the norm of the forward gradient $\nabla \hat{f}$.

**Lemma I.5** (Norm of the Forward Gradient)**.** *Under Assumption I.2, and at the participation rate of $s$, $M$ participating clients training weights $w \in \mathbb{R}^d$ through random perturbations $v \in \mathbb{R}^d$ in* SPRY *derives the accumulated forward gradient $\nabla \hat{f}(w,v;\mathcal{D})$ such that,*

$$\mathbb{E}_{v,\mathcal{D}}||\nabla \hat{f}(w,v;\mathcal{D})||^2 = \mathbb{E}||\nabla f(w)||^2 \left(1 + \left(\frac{2(3d+K-1)}{\widetilde{M}K}\right) \sum_{m\in\mathcal{M}} \sum_{c\in[C]} \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2\right) + \left(\frac{2\sigma_g^2(1-s)(3d+K-1)}{\widetilde{M}K}\right) \sum_{m\in\mathcal{M}} \sum_{c\in[C]} \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2, \tag{27}$$

*where $\mathcal{D}$ is the combination of datasets of all $M$ clients, $K$ is the number of perturbations per batch, $\widetilde{M}$ is the count of clients training a particular weight subset, $\sigma_g^2$ is the upper bound of the global gradient variance. For a total of $C$ classes in $\mathcal{D}$, we define $n_c$ as the sample count of the $c^{th}$ class, $\alpha_c$ is Dirichlet concentration parameter for the $c^{th}$ class, $\mathcal{D}$ is the size of the global data. For a client $m$; $n_{m,c}$ is the sample count of the $c^{th}$ class for client $m$, and $\mathcal{D}_m$ is the size of the data of client $m$.*

*Proof.* The proof follows a similar style of Lemma 2 in MEZO [18]. The difference in our setting is that we have $\nabla \hat{f}(w,v;\mathcal{D})$ which is an aggregate of $\nabla \hat{f}_m(w,v;\mathcal{D})$ derived from the federated clients, while MEZO has results under the setting of a single client.

From Theorem I.4 we have,

$$\mathbb{E}_{v,\mathcal{D}}\left[\nabla \hat{f}(w,v;\mathcal{D})\right]$$

$$= \begin{bmatrix} \nabla f(w_{[1,\frac{d}{M}]}) + \frac{1}{\widetilde{M}} \sum_{m\in\widetilde{\mathcal{M}}_1} \sum_{c=1}^{C} \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right) \mathbb{E}_{(x,y_c)\in\mathcal{D}} \left[\nabla \hat{f}_m(w_{[1,\frac{d}{M}]}, v_{[1,\frac{d}{M}]}; (x,y_c))\right] \\ \nabla f(w_{[\frac{d}{M}+1,\frac{2d}{M}]}) + \frac{1}{\widetilde{M}} \sum_{m\in\widetilde{\mathcal{M}}_2} \sum_{c=1}^{C} \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right) \mathbb{E}_{(x,y_c)\in\mathcal{D}} \left[\nabla \hat{f}_m(w_{[\frac{d}{M}+1,\frac{2d}{M}]}, v_{[\frac{d}{M}+1,\frac{2d}{M}]}; (x,y_c))\right] \\ \vdots \end{bmatrix}^T \tag{28}$$

where

$$\mathbb{E}_{v_{\overline{m}},(x,y_c)\sim\mathcal{D}}\left[\nabla\hat{f}_m(w_{\overline{m}},v_{\overline{m}};(x,y_c))\right] = \frac{1}{|\mathcal{D}|K}\sum_{(x,y_c)\sim\mathcal{D}}\sum_{i\in[K]}\mathbb{E}\left[\left(\nabla f_m\left(w_{\overline{m}};(x,y_c)\right)\cdot v_{i,\overline{m}}\right)v_{i,\overline{m}}^T\right],$$

using Equation 10. The right hand side expectation is also under the randomness of partial perturbations. Here, $K$ is perturbation count, and $|\mathcal{D}|$ is the size of the combined dataset $\mathcal{D} = \cup_{m\in[M]}\mathcal{D}_m$.

To compute the second moment for a client $m$,

$$\begin{aligned}&\mathbb{E}_{v_{\overline{m}},(x,y_c)\sim\mathcal{D}}\left[\nabla\hat{f}_m(w_{\overline{m}},v_{\overline{m}};(x,y_c))\cdot\nabla\hat{f}_m(w_{\overline{m}},v_{\overline{m}};(x,y_c))^T\right]\\ &\quad=\frac{1}{|\mathcal{D}|^2K^2}\sum_{\substack{(x,y_c\sim\mathcal{D})\\(\overline{x},\overline{y}_c\sim\mathcal{D})}}\sum_{\substack{i\in[K]\\j\in[K]}}\mathbb{E}\left[\left(\nabla f_m(w_{\overline{m}};(x,y_c))v_{i,\overline{m}}v_{i,\overline{m}}^T\right)\cdot\left(\nabla f_m(w_{\overline{m}};(x,y_c))^Tv_{j,\overline{m}}v_{j,\overline{m}}^T\right)\right]\end{aligned} \tag{29}$$

For simplicity, let $a$ and $b$ be two arbitrary vectors representing $\mathbb{E}\left[\nabla f_m(w_{\overline{m}};(x,y_c))\right]$ and $\mathbb{E}\left[\nabla f_m(w_{\overline{m}};(\overline{x},\overline{y}_c))^T\right]$, respectively.

For the sum over all perturbations, we have two cases (all the expectations are under the randomness of partial $v$),

1. If $i \neq j$. (Occurs $K(K-1)$ times)

$$\mathbb{E}\left[v_{i,\overline{m}}v_{i,\overline{m}}^T\ ab^T\ v_{j,\overline{m}}v_{j,\overline{m}}^T\right] = \mathbb{E}\left[v_{i,\overline{m}}v_{i,\overline{m}}^T\right]\cdot ab^T\cdot\mathbb{E}\left[v_{j,\overline{m}}v_{j,\overline{m}}^T\right] \tag{30}$$

(since $v_i$ and $v_j$ are independent of each other and $a, b$ don't depend on $v$)

$$= I\cdot ab^T\cdot I = ab^T \tag{31}$$

2. If $i = j$. (Occurs $K$ times)

$$\begin{aligned}\mathbb{E}\left[v_{i,\overline{m}}v_{i,\overline{m}}^T\ ab^T\ v_{j,\overline{m}}v_{j,\overline{m}}^T\right] &= \mathbb{E}\left[v_{i,\overline{m}}v_{i,\overline{m}}^T\ ab^T\ v_{i,\overline{m}}v_{i,\overline{m}}^T\right] &(32)\\ &= \mathbb{E}_{v_i}\left[v_i^4\right]\langle a,b\rangle &(33)\end{aligned}$$

For all such $v_i$ with $i\in[K]$,

$$\begin{aligned}\mathbb{E}_v[v^{\otimes 4}]\langle a,b\rangle &= 3d\ \mathrm{Sym}(I^{\otimes 4})\langle a,b\rangle &(34)\\ &= 2d\cdot ab^T + d\cdot I\cdot a^Tb &(35)\end{aligned}$$

Plugging in the results of the above two cases in Equation 29,

$$\begin{aligned}&\therefore\ \mathbb{E}\left[\nabla\hat{f}_m(w_{\overline{m}},v_{\overline{m}};(x,y_c))\cdot\nabla\hat{f}_m(w_{\overline{m}},v_{\overline{m}};(x,y_c))^T\right]\\ &\quad=\frac{1}{|\mathcal{D}|^2K^2}\sum_{\substack{(x,y_c\sim\mathcal{D})\\(\overline{x},\overline{y}_c\sim\mathcal{D})}}\left[K(K-1)ab^T + 2dKab^T + dK\cdot I\cdot a^Tb\right] &(36)\\ &\quad=\frac{1}{|\mathcal{D}|^2K}\sum_{\substack{(x,y_c\sim\mathcal{D})\\(\overline{x},\overline{y}_c\sim\mathcal{D})}}\Bigg[(2d+K-1)\mathbb{E}\left[\nabla f_m(w_{\overline{m}};(x,y_c))\nabla f_m(w_{\overline{m}};(\overline{x},\overline{y}_c))^T\right]\\ &\qquad\qquad + d\cdot I\cdot\mathbb{E}\Big[\nabla f_m(w_{\overline{m}};(x,y_c))\nabla f_m(w_{\overline{m}};(\overline{x},\overline{y}_c))^T\Big]\Bigg] &(37)\end{aligned}$$

Here, the randomness is under the sampled data.

For the sum over samples of $\mathcal{D}$, we have two cases,

1. If $(x,y_c)\neq(\overline{x},\overline{y}_c)$. (Occurs $|\mathcal{D}|(|\mathcal{D}|-1)$ times)

$$\mathbb{E}[\nabla f_m(w_{\overline{m}};(x,y_c))\nabla f_m(w_{\overline{m}};(\overline{x},\overline{y}_c))^T] = \nabla F_m(w_{\overline{m}})\nabla F_m(w_{\overline{m}})^T \tag{38}$$

2. If $(x, y_c) = (\overline{x}, \overline{y}_c)$. (Occurs $|\mathcal{D}|$ times)

$$\mathbb{E}[\nabla f_m(w_{\overline{m}}; (x, y_c))\nabla f_m(w_{\overline{m}}; (\overline{x}, \overline{y}_c))^T] = \nabla F_m(w_{\overline{m}})\nabla F_m(w_{\overline{m}})^T + \Sigma(w_{\overline{m}}) \tag{39}$$

Combining both the cases, we get

$$\mathbb{E}_{(x,y_c)\sim\mathcal{D}}[\nabla f_m(w_{\overline{m}}; (x, y_c))\nabla f_m(w_{\overline{m}}; (\overline{x}, \overline{y}_c))^T] = |\mathcal{D}|^2\nabla F_m(w_{\overline{m}})\nabla F_m(w_{\overline{m}})^T + |\mathcal{D}| \cdot \Sigma(w_{\overline{m}}) \tag{40}$$

Plugging in Equation 40 in Equation 37,

$$\begin{aligned}
&\mathbb{E}_{v_{\overline{m}},(x,y_c)\sim\mathcal{D}}[\nabla \hat{f}_m(w_{\overline{m}}, v_{\overline{m}}; (x, y_c))\nabla \hat{f}_m(w_{\overline{m}}, v_{\overline{m}}; (x, y_c))^T] \\
&\quad= \frac{1}{|\mathcal{D}|^2 K}\Bigg[(2d + K - 1)|\mathcal{D}|(|\mathcal{D}|\nabla F_m(w_{\overline{m}})\nabla F_m(w_{\overline{m}})^T + \Sigma(w_{\overline{m}})) \\
&\qquad\qquad + d \cdot |\mathcal{D}| \left(|\mathcal{D}| \cdot ||\nabla F_m(w_{\overline{m}})||^2 + \mathrm{tr}\,(\Sigma(w_{\overline{m}}))\right)\Bigg] && (41) \\
&\quad= \frac{(2d + K - 1)}{K}\left(\nabla F_m(w_{\overline{m}})\nabla F_m(w_{\overline{m}})^T + \frac{1}{|\mathcal{D}|}\Sigma(w_{\overline{m}})\right) \\
&\qquad + \frac{d}{K}\left(||\nabla F_m(w_{\overline{m}})||^2 + \frac{1}{|\mathcal{D}|}\mathrm{tr}\,(\Sigma(w_{\overline{m}}))\right) && (42) \\
&\quad= \frac{3d + K - 1}{K}\mathbb{E}_{(x,y_c)\sim\mathcal{D}}||\nabla f_m(w_{\overline{m}}; (x, y_c))||^2 && (43)
\end{aligned}$$

Hence for client $m$, the expected norm of forward gradients under randomness of perturbations $v$ is,

$$\begin{aligned}
\mathbb{E}_{v_{\overline{m}},(x,y_c)\sim\mathcal{D}}||\nabla \hat{f}_m(w_{\overline{m}}, v_{\overline{m}}; (x, y_c))||^2 &= \frac{3d + K - 1}{K}\mathbb{E}_{(x,y_c)\sim\mathcal{D}}||\nabla f_m(w_{\overline{m}}; (x, y_c))||^2 && (44) \\
&= \frac{3d + K - 1}{K}||\nabla F_m(w_{\overline{m}})||^2 && (45)
\end{aligned}$$

For $i^{th}$ row of $\mathbb{E}\left[\nabla \hat{f}(w, v; \mathcal{D})\right]$ of Equation 28,

$$\begin{aligned}
&\mathbb{E}\left|\left|\nabla \hat{f}_i(w_{[\frac{(i-1)d}{M}+1, \frac{id}{M}]}, v_{[\frac{(i-1)d}{M}+1, \frac{id}{M}]}; \mathcal{D})\right|\right|^2 = \mathbb{E}\left|\left|\nabla f_i(w_{[\frac{(i-1)d}{M}+1, \frac{id}{M}]})\right|\right|^2 \\
&+ \frac{1}{(\widetilde{M})^2}\sum_{m\in\widetilde{\mathcal{M}}_i}\sum_{c\in[C]}\left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2 \mathbb{E}_{(x,y_c)\sim\mathcal{D}}\left|\left|\nabla \hat{f}_m(w_{[\frac{(i-1)d}{M}+1, \frac{id}{M}]}, v_{[\frac{(i-1)d}{M}+1, \frac{id}{M}]}; (x, y_c))\right|\right|^2
\end{aligned} \tag{46}$$

The left-hand side expectation under the randomness of sampled data $\mathcal{D}$ and subset of random perturbations $v$.

Plugging in Equation 45 in the above equation and using $\overline{i} = \left[\frac{(i-1)d}{M} + 1, \frac{id}{M}\right]$,

$$\mathbb{E}_{v_{\overline{i}};\mathcal{D}}\left|\left|\nabla \hat{f}_i(w_{\overline{i}}, v_{\overline{i}}; \mathcal{D})\right|\right|^2 = \mathbb{E}\,||\nabla f_i(w_{\overline{i}})||^2 + \frac{(3d + K - 1)}{(\widetilde{M})^2 K}\sum_{m\in\widetilde{\mathcal{M}}_i}||\nabla F_m(w_{\overline{i}})||^2\sum_{c\in[C]}\left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2 \tag{47}$$

Since $\nabla \hat{f}_i, \; \forall \widetilde{\mathcal{M}} \subset \mathcal{M}$ are independent of each other, we can compute the norm of $\nabla \hat{f}$ as follows,

$$\mathbb{E}_{v,\mathcal{D}}||\nabla \hat{f}(w, v; \mathcal{D})||^2 = \mathbb{E}||\nabla f(w)||^2 + \left(\frac{3d + K - 1}{(\widetilde{M})^2 K}\right) \sum_{m\in\mathcal{M}} \left|\left|\nabla F_m(w_{\overline{m}})\right|\right|^2 \sum_{c\in[C]} \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2 \tag{48}$$

$$= \mathbb{E}||\nabla f(w)||^2 + \left(\frac{3d + K - 1}{(\widetilde{M})^2 K}\right) \sum_{m\in\mathcal{M}} \left|\left|\nabla F_m(w_{\overline{m}}) - \nabla f(w_{\overline{m}}) + \nabla f(w_{\overline{m}})\right|\right|^2 \sum_{c\in[C]} \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2 \tag{49}$$

$$= \mathbb{E}||\nabla f(w)||^2 + \left(\frac{2(3d + K - 1)}{(\widetilde{M})^2 K}\right) \sum_{m\in\mathcal{M}} \left|\left|\nabla F_m(w_{\overline{m}}) - \nabla f(w_{\overline{m}})\right|\right|^2 \sum_{c\in[C]} \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2$$
$$+ \left(\frac{2(3d + K - 1)}{(\widetilde{M})^2 K}\right) \sum_{m\in\mathcal{M}} \left|\left|\nabla f(w_{\overline{m}})\right|\right|^2 \sum_{c\in[C]} \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2 \tag{50}$$

Using Assumption I.2 and limited participation rate of $s$,

$$\mathbb{E}_{v,\mathcal{D}}||\nabla \hat{f}(w, v; \mathcal{D})||^2 = \mathbb{E}||\nabla f(w)||^2 + \left(\frac{2\sigma_g^2(1 - s)(3d + K - 1)}{\widetilde{M}K}\right) \sum_{m\in\mathcal{M}} \sum_{c\in[C]} \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2$$
$$+ \left(\frac{2(3d + K - 1)}{\widetilde{M}K}\right) \mathbb{E}\,||\nabla f(w)||^2 \sum_{m\in\mathcal{M}} \sum_{c\in[C]} \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2 \tag{51}$$

Rearranging the terms, we get,

$$\mathbb{E}_{v,\mathcal{D}}||\nabla \hat{f}(w, v; \mathcal{D})||^2 = \mathbb{E}||\nabla f(w)||^2 \left(1 + \left(\frac{2(3d + K - 1)}{\widetilde{M}K}\right) \sum_{m\in\mathcal{M}} \sum_{c\in[C]} \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2\right)$$
$$+ \left(\frac{2\sigma_g^2(1 - s)(3d + K - 1)}{\widetilde{M}K}\right) \sum_{m\in\mathcal{M}} \sum_{c\in[C]} \left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2 \tag{52}$$

□

## I.4 Convergence Rate of SPRY

The general template for the convergence analysis of SPRY is similar to FEDADAM, hence we will follow Theorem 2 of AFO [25]. Our aim here is to highlight the differences in treatment of our gradient estimator $\nabla \hat{f}_m \; \forall m \in [M]$, and the aggregate global gradient $\nabla f$ as shown in Equation 9.

**Theorem I.6.** *Under the assumptions on L-smoothness (Asmp I.1), bounded global variance $\sigma_g^2$ of accumulated gradients (Asmp I.2), and bound on gradient magnitude $G$ (Asmp I.3) and the following conditions on the local learning rate $\eta_\ell$,*

$$\eta_\ell = \min\left\{\mathcal{O}\left(\frac{\tau^2}{\sqrt{\beta_2}\eta GL}\right)^{\frac{1}{2}}, \mathcal{O}\left(\frac{1}{\sqrt{\beta_2}G}\right), \mathcal{O}\left(\frac{\tau^3}{\sqrt{\beta_2}\sqrt{1 - \beta_2}G^2}\right)^{\frac{1}{2}},\right.$$
$$\left.\mathcal{O}\left(\frac{\widetilde{M}K}{\beta_2 G(3d + K - 1)\sum_{m\in[M]}\sum_{c\in[C]}\alpha_{m,c}^2}\right)\right\}; \tag{53}$$

SPRY *satisfies the following bound,*

$$\min_{0\le r\le R} \mathbb{E}_r||\nabla f(w^{(r)})||^2 \le \frac{f(w^{(0)}) - \mathbb{E}_R[f(w^{(R)})]}{\eta R}$$
$$+ \left(2 + \frac{\eta\eta_\ell L}{2\tau^2} + \frac{\sqrt{1 - \beta_2}G\eta_\ell}{\tau^3}\right)\left(\frac{\sigma_g^2(1 - s)(3d + K - 1)}{\widetilde{M}K}\right) \sum_{m\in\mathcal{M}} \sum_{c\in[C]} \alpha_{m,c}^2, \tag{54}$$

*where $R$ is the total round count, $w \in \mathbb{R}^d$ are the trainable weights, $v \in \mathbb{R}^d$ are the random perturbations, $K$ is the count of random perturbations per batch, $\eta$ is the global learning rate, $\tau$ is adaptability hyperparameter, $s$ is client sampling rate. The rest of the symbols are defined in Theorem I.4.*

*Proof.* As shown in Equation 7, an update of the model weights $w$ at server-side in SPRY looks like,

$$w^{(r+1)} = w^{(r)} + \eta \frac{\Delta^{(r)}}{\sqrt{\Lambda^{(r)}} + \tau} \tag{55}$$

Using Assumption I.1 and then the server-side update rule, we have

$$f(w^{(r+1)}) \leq f(w^{(r)}) + \left\langle \nabla f(w^{(r)}), w^{(r+1)} - w^{(r)} \right\rangle + \frac{L}{2} ||w^{(r+1)} - w^{(r)}||^2 \tag{56}$$

$$= f(w^{(r)}) + \eta \left\langle \nabla f(w^{(r)}), \frac{\Delta^{(r)}}{\sqrt{\Lambda^{(r)}} + \tau} \right\rangle + \frac{\eta^2 L}{2} \sum_{i \in [d]} \frac{(\Delta_i^{(r)})^2}{\left(\sqrt{\Lambda_i^{(r)}} + \tau\right)^2} \tag{57}$$

Taking expectation over randomness of round $r$ and simplifying the terms,

$$\mathbb{E}_r[f(w^{(r+1)})] \leq f(w^{(r)}) + \eta \underbrace{\left\langle \nabla f(w^{(r)}), \mathbb{E}_r \left[ \frac{\Delta^{(r)}}{\sqrt{\beta \Lambda^{(r-1)}} + \tau} \right] \right\rangle}_{R_1} + \frac{\eta^2 L}{2} \sum_{i \in [d]} \mathbb{E}_r \left[ \frac{(\Delta_i^{(r)})^2}{(\sqrt{\Lambda_i^{(r)}} + \tau)^2} \right]$$

$$+ \eta \underbrace{\left\langle \nabla f(w^{(r)}), \mathbb{E}_r \left[ \frac{\Delta^{(r)}}{\sqrt{\Lambda^{(r)}} + \tau} - \frac{\Delta^{(r)}}{\sqrt{\beta \Lambda^{(r-1)}} + \tau} \right] \right\rangle}_{R_2} \tag{58}$$

Bounds for $R_2$ are derived in the exactly same manner as "Bounding $R_2$" in Theorem 2 of FEDADAM [25],

$$R_2 \leq \sqrt{1 - \beta_2} \mathbb{E}_r \sum_{j=1}^{d} \frac{G}{\tau} \times \left[ \frac{(\Delta_j^{(r)})^2}{\sqrt{\Lambda_j^{(r)}} + \tau} \right] \tag{59}$$

**Bounding** $R_1$ has a different treatment due to the distinct aggregation strategy of SPRY:

$$R_1 = \left\langle \nabla f(w^{(r)}), \mathbb{E}_r \left[ \frac{\Delta^{(r)}}{\sqrt{\beta \Lambda^{(r-1)}} + \tau} \right] \right\rangle \tag{60}$$

Since $\Delta^{(r)}$ is piece-wise made of aggregations of forward gradients for several parts of the model weights, as shown in Equation 12, we first center each gradient piece for the computation of the squared norm,

$$\therefore R_1 = \left\langle \nabla f(w^{(r)}), \mathbb{E}_r \left[ \frac{-\eta_\ell \nabla \hat{f}(w^{(r)}, v^{(r)}, \mathcal{D})}{\sqrt{\beta \Lambda^{(r-1)}} + \tau} \right] \right\rangle \tag{61}$$

Using $ab \leq (a^2 + b^2)/2$,

$$R_1 \leq -\frac{\eta_\ell}{2} \sum_{j \in [d]} \frac{[\nabla f(w^{(r)})]^2}{\sqrt{\beta \Lambda_j^{(r-1)}} + \tau} + \frac{\eta_\ell}{2} \mathbb{E}_r ||\nabla \hat{f}(w^{(r)}, v^r, \mathcal{D})||^2 \tag{62}$$

Using the result of Lemma I.5,

$$\therefore R_1 \leq -\frac{\eta_\ell}{2} \sum_{j \in [d]} \frac{[\nabla f(w^{(r)})]^2}{\sqrt{\beta \Lambda_j^{(r-1)}} + \tau} + \frac{\eta_\ell}{2} \left( \frac{2\sigma_g^2 (1-s)(3d + K - 1)}{\widetilde{M} K} \right) \sum_{m \in \mathcal{M}} \sum_{c \in [C]} \left( \frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c} \alpha_c}{|\mathcal{D}_m|} \right)^2$$

$$+ \frac{\eta_\ell}{2} \left( 1 + \left( \frac{2(3d + K - 1)}{\widetilde{M} K} \right) \sum_{m \in \mathcal{M}} \sum_{c \in [C]} \left( \frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c} \alpha_c}{|\mathcal{D}_m|} \right)^2 \right) \mathbb{E}_r ||\nabla f(w^{(r)})||^2 \tag{63}$$

Putting $R_1$ and $R_2$ bounds in Equation 58,

$$\begin{aligned}\mathbb{E}_r[f(w^{(r+1)})] \leq f(w^{(r)}) - \frac{\eta\eta_\ell}{2}\sum_{j\in[d]}\frac{[\nabla f(w^{(r)})]^2}{\sqrt{\beta\Lambda_j^{(r-1)}}+\tau} \\ + \frac{\eta\eta_\ell}{2}\left(\frac{2\sigma_g^2(1-s)(3d+K-1)}{\widetilde{M}K}\right)\sum_{m\in\mathcal{M}}\sum_{c\in[C]}\left(\frac{n_c}{|\mathcal{D}|}-\frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2 \\ + \frac{\eta\eta_\ell}{2}\left(1+\left(\frac{2(3d+K-1)}{\widetilde{M}K}\right)\sum_{m\in\mathcal{M}}\sum_{c\in[C]}\left(\frac{n_c}{|\mathcal{D}|}-\frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2\right)\mathbb{E}_r||\nabla f(w^{(r)})||^2 \\ + \frac{\eta^2 L}{2}\sum_{i\in[d]}\mathbb{E}_r\left[\frac{(\Delta_i^{(r)})^2}{\Lambda_i^{(r)}+\tau^2}\right] + \frac{\eta\sqrt{1-\beta_2}G}{\tau}\sum_{i\in[d]}\mathbb{E}_r\left[\frac{(\Delta_i^{(r)})^2}{\sqrt{\Lambda_i^{(r)}}+\tau}\right]\end{aligned} \tag{64}$$

Summing over $r=0$ to $R-1$ and using telescoping sum, we get

$$\begin{aligned}\mathbb{E}_R[f(w^{(R)})] \leq f(w^{(0)}) - \frac{\eta\eta_\ell}{2}\sum_{r=0}^{R-1}\sum_{j\in[d]}\frac{[\nabla f(w^{(r)})]^2}{\sqrt{\beta\Lambda_j^{(r-1)}}+\tau} \\ + \frac{\eta\eta_\ell R}{2}\left(\frac{2\sigma_g^2(1-s)(3d+K-1)}{\widetilde{M}K}\right)\sum_{m\in\mathcal{M}}\sum_{c\in[C]}\left(\frac{n_c}{|\mathcal{D}|}-\frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2 \\ + \frac{\eta\eta_\ell}{2}\left(1+\left(\frac{2(3d+K-1)}{\widetilde{M}K}\right)\sum_{m\in\mathcal{M}}\sum_{c\in[C]}\left(\frac{n_c}{|\mathcal{D}|}-\frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2\right)\sum_{r=0}^{R-1}\mathbb{E}_r||\nabla f(w^{(r)})||^2 \\ + \left(\frac{\eta^2 L}{2}+\frac{\eta\sqrt{1-\beta_2}G}{\tau}\right)\underbrace{\sum_{r=0}^{R-1}\sum_{i\in[d]}\mathbb{E}_r\left[\frac{(\Delta_i^{(r)})^2}{\Lambda_i^{(r)}+\tau^2}\right]}_{R_4}\end{aligned} \tag{65}$$

**Bounding** $R_4$ follows a similar derivation as "Bounding $R_1$",

$$R_4 = \mathbb{E}\sum_{r=0}^{R-1}\sum_{i\in[d]}\frac{(\Delta_i^{(r)})^2}{\Lambda_i^{(r)}+\tau^2} = \mathbb{E}\sum_{r=0}^{R-1}\sum_{i\in[d]}\frac{[-\eta_\ell\nabla\hat{f}(w^{(r)},v^{(r)},\mathcal{D})]_i^2}{\Lambda_i^{(r)}+\tau^2} \tag{66}$$

$$\leq \eta_\ell^2\mathbb{E}\sum_{r=0}^{R-1}\left|\left|\frac{\nabla\hat{f}(w^{(r)},v^{(r)},\mathcal{D})}{\tau^2}\right|\right|^2 \tag{67}$$

Using Lemma I.5 once again,

$$\begin{aligned}\therefore R_4 \leq \frac{\eta_\ell^2}{\tau^2}\left(1+\left(\frac{2(3d+K-1)}{\widetilde{M}K}\right)\sum_{m\in\mathcal{M}}\sum_{c\in[C]}\left(\frac{n_c}{|\mathcal{D}|}-\frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2\right)\mathbb{E}\sum_{r=0}^{R-1}||\nabla f(w^{(r)})||^2 \\ + \frac{\eta_\ell^2}{\tau^2}\left(\frac{2\sigma_g^2(1-s)R(3d+K-1)}{\widetilde{M}K}\right)\sum_{m\in\mathcal{M}}\sum_{c\in[C]}\left(\frac{n_c}{|\mathcal{D}|}-\frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2\end{aligned} \tag{68}$$

Updating Equation 65 with the bounds of $R_4$,

$$\begin{aligned}
\mathbb{E}_R[f(w^{(R)})] \leq f(w^{(0)}) &- \frac{\eta\eta_\ell}{2}\sum_{r=0}^{R-1}\sum_{j\in[d]}\frac{[\nabla f(w^{(r)})]^2}{\sqrt{\beta\Lambda_j^{(r-1)}}+\tau} \\
&+ \frac{\eta\eta_\ell R}{2}\left(\frac{2\sigma_g^2(1-s)(3d+K-1)}{\widetilde{M}K}\right)\sum_{m\in\mathcal{M}}\sum_{c\in[C]}\left(\frac{n_c}{|\mathcal{D}|}-\frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2 \\
&+ \frac{\eta\eta_\ell}{2}\left(1+\left(\frac{2(3d+K-1)}{\widetilde{M}K}\right)\sum_{m\in\mathcal{M}}\sum_{c\in[C]}\left(\frac{n_c}{|\mathcal{D}|}-\frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2\right)\sum_{r=0}^{R-1}\mathbb{E}_r||\nabla f(w^{(r)})||^2 \\
&+ \left(\frac{\eta^2 L}{2}+\frac{\eta\sqrt{1-\beta_2}G}{\tau}\right)\frac{\eta_\ell^2}{\tau^2}\left(1+\left(\frac{2(3d+K-1)}{\widetilde{M}K}\right)\sum_{m\in\mathcal{M}}\sum_{c\in[C]}\left(\frac{n_c}{|\mathcal{D}|}-\frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2\right)\sum_{r=0}^{R-1}\mathbb{E}_r||\nabla f(w^{(r)})||^2 \\
&+ \left(\frac{\eta^2 L}{2}+\frac{\eta\sqrt{1-\beta_2}G}{\tau}\right)\frac{\eta_\ell^2}{\tau^2}\left(\frac{2\sigma_g^2(1-s)R(3d+K-1)}{\widetilde{M}K}\right)\sum_{m\in\mathcal{M}}\sum_{c\in[C]}\left(\frac{n_c}{|\mathcal{D}|}-\frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2
\end{aligned} \tag{69}$$

Rearranging the terms,

$$\begin{aligned}
\sum_{r=0}^{R-1}\sum_{j\in[d]}\frac{[\nabla f(w^{(r)})]^2}{\sqrt{\beta\Lambda_j^{(r-1)}}+\tau} &\leq \frac{f(w^{(0)})-\mathbb{E}_R[f(w^{(R)})]}{\eta} \\
&+ \left(\frac{2R\sigma_g^2(1-s)(3d+K-1)}{\widetilde{M}K}\right)\sum_{m\in\mathcal{M}}\sum_{c\in[C]}\left(\frac{n_c}{|\mathcal{D}|}-\frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2 \\
&+ \left(1+\left(\frac{2(3d+K-1)}{\widetilde{M}K}\right)\sum_{m\in\mathcal{M}}\sum_{c\in[C]}\left(\frac{n_c}{|\mathcal{D}|}-\frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2\right)\sum_{r=0}^{R-1}\mathbb{E}_r||\nabla f(w^{(r)})||^2 \\
&+ \left(\eta L+\frac{2\sqrt{1-\beta_2}G}{\tau}\right)\frac{\eta_\ell}{\tau^2}\left(1+\left(\frac{2(3d+K-1)}{\widetilde{M}K}\right)\sum_{m\in\mathcal{M}}\sum_{c\in[C]}\left(\frac{n_c}{|\mathcal{D}|}-\frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2\right)\sum_{r=0}^{R-1}\mathbb{E}_r||\nabla f(w^{(r)})||^2 \\
&+ \left(\frac{\eta L}{2}+\frac{\sqrt{1-\beta_2}G}{\tau}\right)\frac{\eta_\ell}{\tau^2}\left(\frac{R\sigma_g^2(1-s)(3d+K-1)}{\widetilde{M}K}\right)\sum_{m\in\mathcal{M}}\sum_{c\in[C]}\left(\frac{n_c}{|\mathcal{D}|}-\frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2
\end{aligned} \tag{70}$$

Getting a lower bound for the left hand side term through using the fact $\sqrt{\Lambda^{(r-1)}} \leq \eta_\ell KG$ from Theorem 2 of AFO [25],

$$\sum_{r=0}^{R-1}\sum_{j=1}^{d}\frac{\mathbb{E}_r[\nabla f(w^{(r)})]_j^2}{\sqrt{\beta_2\Lambda_j^{(r-1)}}+\tau} \geq \sum_{r=0}^{R-1}\sum_{j=1}^{d}\frac{\mathbb{E}_r[\nabla f(w^{(r)})]_j^2}{\sqrt{\beta_2}\eta_\ell KG+\tau} \geq \frac{R}{\sqrt{\beta_2}\eta_\ell KG+\tau}\min_{0\leq r\leq R}\mathbb{E}_r||\nabla f(w^{(r)})||^2 \tag{71}$$

$$\therefore \frac{R}{\sqrt{\beta_2}\eta_\ell KG + \tau} \min_{0 \le r \le R} \mathbb{E}_r ||\nabla f(w^{(r)})||^2 \le \frac{f(w^{(0)}) - \mathbb{E}_R[f(w^{(R)})]}{\eta}$$
$$+ \left(2 + \frac{\eta\eta_\ell L}{2\tau^2} + \frac{\sqrt{1-\beta_2}G\eta_\ell}{\tau^3}\right)\left(\frac{R\sigma_g^2(1-s)(3d+K-1)}{\widetilde{M}K}\right)\sum_{m\in\mathcal{M}}\sum_{c\in[C]}\left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2$$
$$+ \left(1 + \left(\frac{2(3d+K-1)}{\widetilde{M}K}\right)\sum_{m\in\mathcal{M}}\sum_{c\in[C]}\left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2\right)\mathbb{E}\sum_{r=0}^{R-1}||\nabla f(w^{(r)})||^2$$
$$+ \left(\eta L + \frac{2\sqrt{1-\beta_2}G}{\tau}\right)\frac{\eta_\ell}{\tau^2}\left(1 + \left(\frac{2(3d+K-1)}{\widetilde{M}K}\right)\sum_{m\in\mathcal{M}}\sum_{c\in[C]}\left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2\right)\mathbb{E}\sum_{r=0}^{R-1}||\nabla f(w^{(r)})||^2 \tag{72}$$

Considering the coefficients of $\mathbb{E}||\nabla f(w^{(r)})||^2$ terms, conditioning on the following inequality,

$$\frac{R}{\sqrt{\beta_2}\eta_\ell KG + \tau} \ge \left(\frac{\eta\eta_\ell L}{\tau^2} + \frac{2\eta_\ell\sqrt{1-\beta_2}G}{\tau^3} + 1\right)\left(1 + \frac{2(3d+K-1)}{\widetilde{M}K}\sum_{m\in[M]}\sum_{c\in[C]}\left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2\right) \tag{73}$$

We get the following condition on the local learning rate,

$$\eta_\ell = \min\Bigg\{\mathcal{O}\left(\frac{\tau^2}{\sqrt{\beta_2}\eta GL}\right)^{\frac{1}{2}}, \mathcal{O}\left(\frac{1}{\sqrt{\beta_2}G}\right), \mathcal{O}\left(\frac{\tau^3}{\sqrt{\beta_2}\sqrt{1-\beta_2}G^2}\right)^{\frac{1}{2}},$$
$$\mathcal{O}\left(\frac{\widetilde{M}K}{\beta_2 G(3d+K-1)\sum_{m\in[M]}\sum_{c\in[C]}\left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2}\right)\Bigg\} \tag{74}$$

for the following bound on the gradient norm,

$$\min_{0\le r\le R}\mathbb{E}_r||\nabla f(w^{(r)})||^2 \le \frac{f(w^{(0)}) - \mathbb{E}_R[f(w^{(R)})]}{\eta R}$$
$$+ \left(2 + \frac{\eta\eta_\ell L}{2\tau^2} + \frac{\sqrt{1-\beta_2}G\eta_\ell}{\tau^3}\right)\left(\frac{\sigma_g^2(1-s)(3d+K-1)}{\widetilde{M}K}\right)\sum_{m\in\mathcal{M}}\sum_{c\in[C]}\left(\frac{n_c}{|\mathcal{D}|} - \frac{n_{m,c}\alpha_c}{|\mathcal{D}_m|}\right)^2 \tag{75}$$

□