A Frequently Asked Questions

> Why can't clients just remember their local parameters? Why can't we use a stateful algorithm?

In real-world cross-device FL settings, the population size is huge (*e.g.,* millions of clients), only a small number of clients participate in each round (*e.g.,* 200 clients), and a client usually participates **at most once** during the entire training process. Under this setting, algorithms cannot rely on client state, such as local parameters, from a previous round because in almost every case it will not exist. In fact, stateful algorithms result in performance degradation in the cross-device setting due to state getting "stale" between rounds (see the SCAFFOLD discussion in Reddi et al. [48] Sec. 5.1). We empirically observe a performance difference in Table 1, where FEDAVG (which performs identically to the stateful FEDPER approach of Arivazhagan et al. [4] for this task) does not match the performance of FEDRECON for matrix factorization. Additionally, in partially local FL, a stateful algorithm would result in all non-sampled clients being without trained local parameters. In the real-world setting from Section 7, this would mean more than **99%** of clients would not have working models, preventing practical deployment. Note that clients can still optionally store final local variables for inference after training as mentioned in Section 4.1, since this is independent of any federated process.

> Why perform reconstruction of local parameters? Isn't reconstruction wasteful?

Since clients are very unlikely to be reached repeatedly by large-scale cross-device training (see above answer), in practice this cost is nearly the same as the cost of initializing these local parameters and training them with stateful clients. Additionally, reconstruction provides a natural way for new clients unseen during training to produce their own partially local models offline (see Section 4.1)–without this step, the vast majority of clients in the real-world deployment described in Section 7 would not be able to use the model. Reconstruction also ensures local parameters are always fresh, avoiding staleness issues (see above answer). Reconstruction also allows us to save on communication cost compared to fully global models, which is typically more of a limiting resource in federated learning than local client computation [34, 38]. See the improvement in the accuracy-communication tradeoff in Table 2 and Figure 2. Lastly, in Section 4.2 and Appendix B we provide intuitive and theoretical arguments for why reconstruction naturally leads to training global parameters that can be easily used to train personal local parameters, similar to the existing intuition around Model-Agnostic Meta Learning methods [19]. In Figure 3 and Table 1, we also empirically demonstrate that reconstruction leads to performant final models with minimal gradient steps (even on unseen clients).

> How is this different from *Paper X*?

Our work differs from other work in that (1) it addresses partially local federated learning and (2) it proposes a stateless algorithm for this setting, making partially local federated learning practical in large-scale cross-device settings. Many other federated learning works either don't address (1) *e.g.,* because they do not involve models where some (privacy-sensitive) parameters are local and some parameters are global, or don't address (2) *e.g.,* because they introduce algorithms that require stateful clients. Our method has also been extensively validated through simulation experiments using Stack Overflow and MovieLens user data and a real-world deployment to a cross-device setting with hundreds of millions of clients (rare in FL papers). We have also open-sourced the code that was deployed, making practical partially local federated learning widely available; we know of no other work that has done this. We discuss specific differences from previous works in Section 2.

> How do we know the approach converges?

We show that our approach converges empirically in Figures 2, 4 and 5. We also show in Table 1 that we achieve better results on unseen users than a **server-trained** matrix factorization model, again demonstrating empirical performance. We also show a theoretical connection to MAML in Appendix B, which justifies the fast reconstruction in Figure 3. While we do not present additional theoretical convergence results, one of our core contributions in this work is our deployment of this approach at scale in a real-world setting with hundreds of millions of clients, improving recommendation CTR by 29.3% (see Section 7). We believe that this contribution does more to validate the approach than theoretical convergence results.

> How do we split the model into global and local parts in partially local FL?

If the model has user-specific or very privacy-sensitive parameters (as in the matrix factorization case), then these parameters can naturally be local parameters. For other models, the global-local split follows from the use-case and communication requirements. If communication is a concern, making more variables local may be a way of reducing communication cost. As an example, in Section 5.1.2 we motivate the next word prediction use-case, where having a partially local model with local OOV embeddings can be useful for handling diverse client inputs while reducing communication.

B Proof of Connection to Meta Learning

As described in Section 4.2, our meta-parameters are g and our task-specific parameters are l_i for client i. We want to find g minimizing the objective:

$$\mathbb{E}_{i\sim\mathcal{P}}\, f_i(g \parallel l_i) = \mathbb{E}_{i\sim\mathcal{P}}\, f_i(g \parallel R(\mathcal{D}_{i,s}, \mathcal{L}, g)] \tag{3}$$

where $g \parallel l_i$ denotes the concatenation of g and l_i and $f_i(g \parallel l_i) = \mathbb{E}_{\xi\in\mathcal{D}_{i,q}}[f_i(g \parallel l_i, \xi)]$.

We now show that the instantiation of our framework where R performs $k_r \geq 1$ steps of gradient descent on initialized local parameters using $\mathcal{D}_{i,s}$ and U performs $k_u = 1$ step of gradient descent using $\mathcal{D}_{i,q}$ is *already* minimizing the first-order terms in this objective (*i.e.,* this version of FEDRECON is performing first-order meta learning).

Taking the server update from Algorithm 1 for round t, we have:

$$g^{(t+1)} \leftarrow g^{(t)} + \eta_s \sum_{i\in\mathcal{S}^{(t)}} \frac{n_i}{n}\Delta_i^{(t)}$$

Given that U performs $k_u = 1$ step of SGD with learning rate η_u, the right side is equivalent to:

$$g^{(t)} + \eta_s \sum_{i\in\mathcal{S}^{(t)}} \frac{n_i}{n}((g^{(t)} - \eta_u \nabla_g f_i(g^{(t)} \parallel l_i^{(t)})) - g^{(t)})$$

Simplifying and writing $l_i^{(t)}$ in terms of R:

$$g^{(t)} - \eta_s\eta_u \sum_{i\in\mathcal{S}^{(t)}} \frac{n_i}{n}(\nabla_g f_i(g^{(t)} \parallel R(\mathcal{D}_{i,s}, \mathcal{L}, g^{(t)})))$$

Note that this corresponds to sampling a batch of tasks and performing SGD on the global parameters in Equation (3) with learning rate $\eta_s\eta_u$, weighted by the number of examples by task (this can be omitted if desired). An important detail is that the result of R is treated as a constant in computing $\nabla_g f_i$. We argue that this corresponds to neglecting the second-order terms in the gradient. Since R performs k_r steps of SGD, we can write out each step:

$$\begin{aligned}
l_i^{(t,1)} &= l_i^{(t,0)} - \eta_r \nabla_{l_i} f_i(g^{(t)} \parallel l_i^{(t,0)}) \\
l_i^{(t,2)} &= l_i^{(t,1)} - \eta_r \nabla_{l_i} f_i(g^{(t)} \parallel l_i^{(t,1)}) \\
&\dots \\
l_i^{(t)} = l_i^{(t,k_r)} &= l_i^{(t,k_r-1)} - \eta_r \nabla_{l_i} f_i(g^{(t)} \parallel l_i^{(t,k_r-1)})
\end{aligned}$$

where $l_i^{(t,0)}$ is a random initialization of the parameters in $\mathcal{L}$. So using the chain rule $\nabla_g f_i(g^{(t)} \parallel R(\mathcal{D}_{i,s}, \mathcal{L}, g^{(t)}))$ can be written as:

$$\begin{aligned}
&\nabla_g f_i(g^{(t)} \parallel l_i^{(t,k_r)}) + \nabla_{l_i} f_i(g^{(t)} \parallel l_i^{(t,k_r)})\nabla_g \left(l_i^{(t,0)} - \eta_r \sum_{j=0}^{k_r-1} \nabla_{l_i} f_i(g^{(t)} \parallel l_i^{(t,j)}) \right) \\
&= \nabla_g f_i(g^{(t)} \parallel l_i^{(t,k_r)}) - \eta_r \nabla_{l_i} f_i(g^{(t)} \parallel l_i^{(t,k_r)}) \sum_{j=0}^{k_r-1} \nabla_g \nabla_{l_i} f_i(g^{(t)} \parallel l_i^{(t,j)})
\end{aligned}$$

Treating the output of R as constant in g therefore corresponds to dropping the Jacobian-gradient terms (which are second-order partial derivatives in elements of g and l_i), leaving us with $\nabla_g f_i(g^{(t)} \parallel l_i^{(t,k_r)})$. Using the full matrix of these terms would lead to additional computational cost and memory cost quadratic in the number of local parameters, which is generally impractical on heterogeneous client devices. Neglecting similar terms (these terms reduce to the Hessian of f_i in the case that $g = l_i$) has been shown to cause minimal drop in performance of meta learning algorithms [19, 46].

We have shown FEDRECON trains global parameters g for fast reconstruction of local parameters l, enabling partially local federated learning without requiring clients to maintain state. In Section 5.2 we observe that our method empirically produces g more conducive to fast, performant reconstruction on unseen clients than standard centralized or federated training (*e.g.,* see SERVER+RECONEVAL vs. FEDRECON in Table 1). We see in Figure 3 that *just one reconstruction step* is sufficient to recover the majority of performance.

C Datasets, Models, and Hyperparameters

Below we provide further detail on our evaluation tasks, including descriptions of datasets, models, and hyperparameters. Our open-source framework described in Section 6 contains commands to reproduce these tasks.

C.1 Matrix Factorization

We evaluate on federated matrix factorization using the popular MovieLens 1M collaborative filtering dataset [29]. The dataset consists of 1,000,209 ratings on 3,706 movies from 6,040 users who joined MovieLens in 2000.[5] The dataset is licensed for research use [25].

We perform two kinds of evaluation: (1) standard (federated) evaluation on seen users and (2) RECONEVAL on *unseen* users. For (1), we split each user's ratings into 80% train, 10% validation, and 10% test by timestamp. This is the typical evaluation setup in the matrix factorization literature [30]. For (2) we split the users randomly into 80% train, 10% validation, and 10% test; we train with the train users and report results on test users. This setup tests the model's ability to generalize to unseen users without trained user embeddings. Note that without reconstruction of user embeddings, we would expect the model to perform poorly on unseen users with randomly initialized user embeddings, and this is what we observe in Table 3 (compare CENTRALIZED + STANDARDEVAL (UNSEEN) and CENTRALIZED + RECONEVAL).

The model learns P and Q such that $R \approx PQ^\top$ as discussed in Section 3, with embedding dimensionality $K = 50$. We report root-mean-square-error (RMSE) and rating prediction accuracy (rounding predicted ratings to the nearest integer, how often does the model predict the correct rating?).

We apply FEDRECON with local user embeddings P_u and global item matrix Q. The dataset split function S splits each user's data into half for the support and query sets (see Appendix C.2 for a discussion of modifying this). R and U each perform up to 50 gradient descent steps. We tried smaller numbers of steps in Figure 3, and found that results mostly plateaued around 10 steps and stopped improving at 50 steps. We use a batch size of 5 for federated training. We grid over server learning rate $\eta_s \in [0.1, 0.5, 1.0]$, reconstruction learning rate $\eta_r \in [0.1, 0.5]$, and client update learning rate $\eta_u \in [0.1, 0.5]$. Note that η_u corresponds to the client learning rate in the generalized FEDAVG presented by Reddi et al. [48]. η_r is newly introduced by our method, and we have found that setting it to the same as the client learning rate or slightly lower works well across tasks. We run 500 rounds of training with 100 clients randomly selected per round. For centralized baselines, we run 20 epochs of training with a batch size of 300. We report the configuration with best final validation performance for each setting. We rerun experiments 3x and report average metrics.

C.2 Next Word Prediction

We use the federated Stack Overflow dataset introduced in TensorFlow [51]. The dataset consists of public questions and answers from Stack Overflow, naturally partitioned into clients by posts from different users on the site. The dataset has 342,477 training clients with 135,818,730 examples and 38,758 held-out clients with 16,491,230 examples. The dataset is licensed under the Creative Commons Attribution-ShareAlike 3.0 Unported License [12].

---

[5] Data was provided voluntarily by users. The full dataset includes some demographic information for these users, *e.g.,* their gender, but we do not use this information in this work.

Table 3: Movielens matrix factorization root-mean-square-error (lower is better) and rating prediction accuracy (higher is better). STANDARD EVAL is on seen users, RECONEVAL is on held-out users. Results within 2% of best for each metric are in bold.

	RMSE ↓	ACCURACY ↑
CENTRALIZED + STANDARD EVAL	.923	43.2
CENTRALIZED + STANDARD EVAL (UNSEEN)	3.80	0.0
CENTRALIZED + RECONEVAL	1.36	40.8
FEDRECON (OURS)	.907	**43.3**
FEDRECON (NO SPLIT)	.912	42.5
FEDRECON (JOINT TRAINING)	.915	42.1
FEDRECON (ADAGRAD)	**.883**	**44.1**

Table 4: Stack Overflow next word prediction accuracy and communication per round, per client. FEDYOGI and OOV/FULL FINETUNING require communication of all model parameters, FEDRECON does not (see Figure 2). Results within 2% of best for each vocabulary size are in bold.

VOCAB. SIZE	1K	5K	10K	COMMUNICATION
FEDRECON (1 OOV)	24.1	26.2	26.4	$2\|g\|$
FEDRECON (500 OOV)	**29.6**	**28.1**	**27.7**	$2\|g\|$
FEDRECON (500 OOV, NO SPLIT)	28.8	**28.1**	**27.7**	$2\|g\|$
FEDRECON (500 OOV, JOINT TRAINING)	**29.3**	**27.9**	**27.7**	$2\|g\|$

We perform a autoregressive next word prediction task, predicting the next word in a sentence given the previous words. For ease of comparison, we process the data and use the model as described in (Reddi et al. [48] [Appendix C.4]); we restrict the vocabulary to one of [1000, 5000, 10000] most popular words and use padding and truncation to ensure each sentence has exactly 20 words. We also restrict each client to have at most 1000 sentences. We use the same LSTM with input embedding dimension 96 and output dimension 670, except we enable a variable number of OOV embeddings as motivated in Section 5.1.2. We compare across different numbers of OOV embeddings in Figure 6 and observed that performance plateaus for all vocabulary sizes by 500 OOV, so we use this for experiments with multiple OOV embeddings. Just as Reddi et al. [48], we use a local batch size of 16. We also use Yogi for the server update step. We report the top-1 accuracy, ignoring special tokens representing the out-of-vocabulary tokens, padding, and beginning/end of sentences.

We apply FEDRECON where the OOV embeddings are local and the rest of the model (including the core vocabulary embeddings) is global. The dataset split function S splits each user's data in half by timestamp–we found that we achieved similar performance if the split was not half, as long as the split occurred by timestamp (otherwise, support and query examples are too similar and results are unrealistically inflated). R and U each perform up to 100 gradient steps. We tried smaller numbers of steps in Figure 3, and found that results mostly plateaued around 10 steps and stopped improving at 100 steps. We perform 2500 rounds of training with 200 clients randomly selected per round. We grid over server learning rate $\eta_s \in [0.01, 0.1, 0.3]$, reconstruction learning rate $\eta_r \in [0.1, 0.3]$, and client update learning rate $\eta_u \in [0.1, 0.3]$, similar to the grid used by Reddi et al. [48] and applying the heuristic that reconstruction learning rate can be set to about the same as client learning rate. Where applicable, we use the same hyperparameters and learning rate grids for the FEDAVG baselines. We rerun experiments 3x and report average held-out accuracy for the best configuration for each setting.

D Additional Empirical Results

In Tables 3 and 4 we present experimental results for FEDRECON on matrix factorization and next word prediction tasks, in addition to the key results discussed in Section 5.2. In these tables, we include some previously discussed results for ease of comparison. We call out interesting comparisons and additional figures in the discussion below.

Sharing Support and Query Data: If clients have very limited data (*e.g.,* some Stack Overflow clients have just one example) partitioning the data into disjoint support and query sets may be undesirable. As an ablation, in Tables 3 and 4 we evaluate FEDRECON (NO SPLIT), where the full

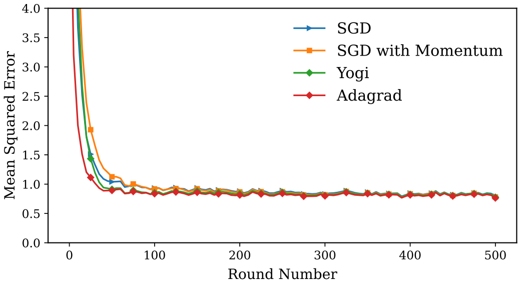

Figure 4: MovieLens matrix factorization mean squared error (MSE) loss for FEDRECON across different server optimizers.

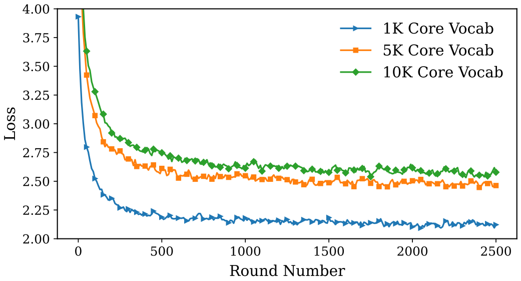

Figure 5: Stack Overflow next word prediction cross-entropy loss for FEDRECON with varying core vocabulary size, with 500 out-of-vocabulary embeddings.

client dataset is used for both support and query (keeping RECONEVAL the same for fairness); we see that we can relax the requirement for these sets to be disjoint with minimal drop in performance across MovieLens and Stack Overflow. We make use of this in our real-world deployment described in Section 7, where it also helps combat data sparsity.

Joint Training after Reconstruction: FEDRECON involves alternating between (1) training local parameters with global parameters frozen during reconstruction and (2) training global parameters with local parameters frozen after reconstruction. We also tried joint training after reconstruction, where we update local and global parameters concurrently after reconstruction. In Tables 3 and 4 we evaluate FEDRECON (JOINT TRAINING). We see similar but slightly degraded performance in this setting. This suggests that freezing local parameters during (2) is useful for ensuring that global parameters are updated significantly (otherwise training does not progress across rounds since local parameters are not aggregated).

Adaptive Optimizers: FEDRECON treats aggregated global parameter updates as an "antigradient" that can be input into different server optimizers, building off of Reddi et al. [48]. In Figure 4 we observe that we can apply different server optimizers to the MovieLens matrix factorization task and loss converges well. In Table 3 we report an improved result for FEDRECON using ADAGRAD. All Stack Overflow results use Yogi as the server optimizer for consistency with Reddi et al. [48]–we also observed this improved performance over SGD. These results indicate that FEDRECON can be profitably combined with other advances in federated optimization.

Stack Overflow Vocabulary and OOV Sizes: In Figure 5 we plot loss over rounds for different core vocabulary sizes for the Stack Overflow next word prediction task (applying FEDRECON with 500 OOV embeddings fixed). We observe that losses are lower as core vocabulary coverage decreases, consistent with the hypothesis posited in Section 5.2 that lowering the size of the core vocabulary provides more "training data" for the local OOV embeddings, which improves local personalization performance (note that the same trend does not occur for the FEDYOGI result from Table 2).

In Figure 6 we see that adding OOV buckets improves accuracy the most for a smaller core vocabulary, as expected. For all vocabulary sizes, performance plateaus around 500 OOV buckets.

MovieLens Centralized Evaluation on Unseen Users: In Table 3 we evaluate a standard server-trained matrix factorization model on unseen users in two ways: (1) standard evaluation, where we randomly initialize a local embedding for the user and compute metrics, and (2) RECONEVAL, where we split each user's data, reconstruct the user embedding using support data, and compute

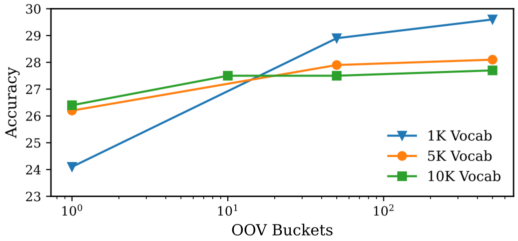

Figure 6: Stack Overflow next word prediction accuracy for FedRecon with varying numbers of out-of-vocabulary buckets for each core vocabulary size.

metrics on query data. Note that since unseen users do not have any trained user embeddings, these initialization strategies are needed for evaluation. (1) produces bad results as expected given the randomly initialized user embeddings, indicating that ReconEval is needed to produce reasonable results. As discussed in Section 5.2, FedRecon produces improved results with ReconEval on unseen users compared to centralized training because it trains global parameters conducive to reconstruction.

E Privacy Implications and Limitations

FedRecon enables clients to learn models without sending privacy-sensitive parameters to a central server, as in the matrix factorization application. More broadly, even for use-cases without user-specific parameters, our method also provides a practical alternative to centralized training (and centralized data collection) and fully global federated learning, with some of the benefits of fully local learning. We hope that our contributions push future machine learning applications towards requiring less centralized data collection and less communication of privacy-sensitive personal information; this is why we have open-sourced the code framework used to deploy FedRecon in a large-scale real-world federated learning application (see Section 7), making practical partially local federated learning widely available.

However, like other federated learning algorithms, the method involves clients sending gradient updates to be aggregated on a central server. Several works have shown that this can lead to leakage of client information to a curious server [23, 53, 57, 59]. These attacks are generally most successful in simple cases, *e.g.,* if each client has just one training example used to derive their gradient and performs one gradient step. Still, significant leakage and even reconstruction of training data has been observed in more realistic settings [23, 57].

Vanilla FedRecon may naturally provide a degree of protection against gradient leakage attacks. Since only a subset of model parameters' updates are communicated to the server, and these updates are directly calculated using only the query subset of the data, the attacks introduced in previous work may not be applicable. However, further attacks could be developed and vanilla FedRecon (as well as other federated learning algorithms) provides no formal guarantee that gradients for global parameters will not leak information. Thus, in the most privacy-sensitive applications, FedRecon may be augmented with differential privacy [17] or secure aggregation [8] applied to global parameter updates, to provide provable guarantees about the information the server receives. Below we describe why applying differential privacy may be particularly promising for FedRecon.

E.1 Application to Differential Privacy

Federated learning has provided a natural application for differentially private computing [17, 24]. Differential privacy is typically applied in (federated) machine learning via variants of DP-SGD, as outlined by Abadi et al. [1] and McMahan et al. [45]. This algorithm clips and noises gradients in the spherical geometry of ℓ^2 at each step of model training, introducing a dependence on model dimensionality. For an iterative procedure whose intermediate results are vectors in $\mathbb{R}^d$, each iteration must add noise at the scale of $\sqrt{d}$; this $\sqrt{d}$ subsequently appears as a steady-state risk in regret-based analysis of differentially private optimization; see *e.g.,* Bassily et al. [6], Wu et al. [55], Bassily et al. [5].

FEDRECON provides a natural parameter to improve the performance of differentially private federated training: the dimensionality of the global parameters. By splitting the model into local and global portions, FEDRECON reduces the dimensionality of aggregated parameters, reducing the variance of the noise which must be added to ensure user-level differential privacy [45]. We emphasize that this is a systems-focused application of FEDRECON, rather than a fundamentally new differential privacy algorithm; tuning the number of global parameters composes rather than competes with algorithmic advances in differential privacy. We believe this is an interesting area for future work.