FEATHERS: Federated Architecture and Hyperparameter Search

Jonas Seng[1] **Pooja Prasad**[2] **Devendra Singh Dhami**[2] **Martin Mundt**[1] **Kristian Kersting**[1,3,4,5]

[1]Technical University Darmstadt
[2]Technical University Eindhoven
[3]hessian.ai
[4]Centre of Cognitive Science TU Darmstadt
[5]German Research Centre AI (DFKI)

Abstract Deep neural architectures have a profound impact on achieved performance in many of today's AI tasks, yet their design still heavily relies on human prior knowledge and experience. Neural architecture search (NAS) together with hyperparameter optimization (HO) helps to reduce this dependence. However, state-of-the-art NAS and HO rapidly become infeasible with increasing amounts of data being stored in a distributed fashion. This is mainly because these methods are not designed for distributed environments and typically violate data privacy regulations such as GDPR and CCPA. As a remedy, we introduce FEATHERS—**FE**derated **A**rchi**T**ecture and **H**yp**ER**parameter **S**earch, a method that not only optimizes both neural architectures *and* optimization-related hyperparameters jointly in distributed data settings, but further provably preserves data privacy through the use of differential privacy (DP). We show that FEATHERS efficiently optimizes architectural and optimization-related hyperparameters alike while demonstrating convergence on classification tasks at no detriment to model performance when complying with privacy constraints.

1 Introduction

Federated learning (FL) is a distributed machine learning paradigm aiming to learn a shared model on data distributed at different locations without ever exchanging the data itself (McMahan et al., 2017), rendering it a promising way to leverage ML in industries with high privacy standards (e.g. healthcare). As in classical machine learning (ML), neural architectures and other hyperparameters have to be set in FL before training. Although increasingly more data is stored in decentralized systems and privacy awareness is rising, most NAS/HO approaches are still tailored to classical, i.e. centralized, ML settings. (Kairouz et al., 2021; Zoph and Le, 2017; Pham et al., 2018; Liu et al., 2019; Agrawal et al., 2021). Recent approaches tackle NAS and HO in federated environments (He et al., 2020a; Khodak et al., 2021; Singh et al., 2020; Zhu and Jin, 2021). However, they optimize neural architectures *or* hyperparameters, thus disregarding inherent dependence between hyperparameters and neural architectures (Zela et al., 2018). Performing NAS and HO sequentially as a naive solution is often infeasible in FL due to high communication costs since models are exchanged several times during learning. The regular model exchange uncovers another shortcoming: Model- and hyperparameters carry sensitive information about the training data (Fredrikson et al., 2015; Papernot and Steinke, 2022; Huang et al., 2022) or can be poisoned (Ye et al., 2022). However, existing federated NAS/HO do not provide privacy-preserving mechanisms

Method	NAS	HO	DP	Fed.
DARTS (Liu et al., 2019)	✓	✗	✗	✗
DP-FNAS (Singh et al., 2020)	✓	✗	✓	✓
FedNAS (He et al., 2020a)	✓	✗	✗	✓
DP-FTS-DE (Dai et al., 2021)	✗	✓	✓	✓
FedEx (Khodak et al., 2021)	✗	✓	✗	✓
RTFedNAS (Zhu and Jin, 2021)	✓	✗	✗	✓
FEATHERS	✓	✓	✓	✓

Table 1: We compare the capabilities of FEATHERS with the capabilities of other recent approaches on federated HO/NAS.

to protect sensitive information. To address these challenges, we propose **FEATHERS** – **FE**derated **A**rchi**T**ecture and **H**yp**ER**parameter **S**earch, a novel method that synergizes differentiable NAS and bandit-inspired HO in an alternating optimization process, thus optimizing architectures and hyperparameters *jointly*, while enabling privacy-preserving federated learning (see Fig. 1, Tab. 1).

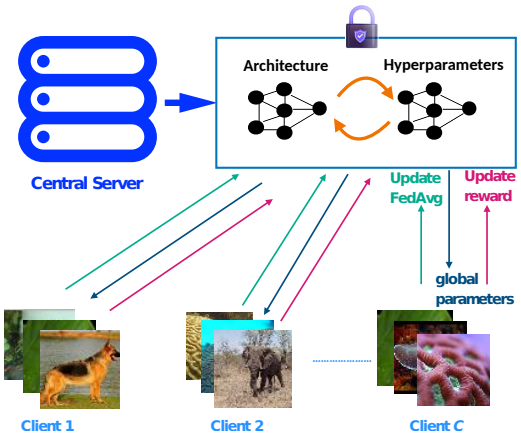

Figure 1: FEATHERS jointly optimizes neural architectures and hyperparameters in FL while providing privacy guarantees. Differentiable NAS paired with FedAvg enables efficient federated architecture search while bandit-based HO optimitzes hyperparameters.

Overall, we make the following contributions: **(1)** We propose a novel method, FEATHERS, that jointly optimizes neural architectures and hyperparameters in distributed data settings. **(2)** Further, we prove convergence properties of the HO and NAS phase of FEATHERS and **(3)** provide and prove privacy guarantees during the search- and evaluation stage. **(4)** We empirically show that FEATHERS finds well-suited architectures and hyperparameters on various classification tasks.

2 Related Work

To accommodate the increasing amount of data being stored in distributed systems and increasing privacy requirements, in FL, a set of clients aim to collaboratively learn a shared global model without exchanging training data (McMahan et al., 2017). Instead, clients train a local private copy of the global model, whose parameters are updated regularly via parameter aggregation. Like in classical ML, hyperparameter optimization (HO) algorithms have been developed for FL to obtain high-performing models automatically. For example, FLoRA (Zhou et al., 2021) and FedEx (Khodak et al., 2021) view the HO problem as a bandit problem and find hyperparameter configurations by optimizing a reward function indicating the performance of configurations. Other approaches adapt hyperparameters during FL using gradient information (Koskela and Honkela, 2018), reinforcement learning (Mostafa, 2019), evolutionary search (Agrawal et al., 2021). In FL, neural networks are widely used, thus performing federated neural architecture search (NAS) is increasingly important to obtain state-of-the-art models. Since differentiable-based NAS methods naturally allow for parameter averaging commonly used in FL, prominent approaches leverage such algorithms for federated NAS (Zhu et al., 2021; He et al., 2020a,b; Singh et al., 2020).

A pressing concern in FL is the protection of private information since models are shared regularly and both model parameters of neural networks and hyperparameters can carry private information of the training data (Fredrikson et al., 2015; Papernot and Steinke, 2022; Huang et al., 2022). Differential Privacy (DP) (Dwork, 2006) protects sensitive information in data via randomization of query outputs and has been successfully employed in FL to avoid privacy leakage (Abadi et al., 2016; Dai et al., 2021).

3 FEATHERS – Joint NAS and HO under Privacy Guarantees

Consider a FL setting with C clients $\mathcal{C}$, each holding a dataset $D_1, ..., D_C$ that is split into training $\langle \mathbf{X}^{(c)}_{train}, \mathbf{y}^{(c)}_{train} \rangle$ and validation data $\langle \mathbf{X}^{(c)}_{val}, \mathbf{y}^{(c)}_{val} \rangle$. We aim to find an architecture $\mathbf{a} \in \mathcal{A}$ and hyperparameters $\mathbf{h} \in \mathcal{H}$ minimizing the global validation loss, i.e., to optimize

$$\min_{\mathbf{a},\mathbf{h}} \sum_{c \in \mathcal{C}} v_c \cdot \mathcal{L}_{\mathbf{a},\mathbf{h}}(\mathbf{w}^*, \mathbf{X}^{(c)}_{val}, \mathbf{y}^{(c)}_{val}) \quad \text{with} \quad \mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_{c \in \mathcal{C}} v_c \cdot \mathcal{L}_{\mathbf{a},\mathbf{h}}(\mathbf{w}, \mathbf{X}^{(c)}_{train}, \mathbf{y}^{(c)}_{train}). \tag{1}$$

Here, $\mathbf{w} \in \mathbb{R}^n$ refers to the parameters of a neural network (model parameters) and $v_c := \frac{|\mathbf{X}_{val}^{(c)}|}{\sum_{c \in \mathcal{C}} |\mathbf{X}_{val}^{(c)}|}$ is the weight of a client c, i.e., how much the local validation loss of c contributes to the global validation loss. We denote the global training- and validation loss as $\mathcal{L}_{\mathbf{a},\mathbf{h}}(\mathbf{w}, \mathbf{X}_{train}, \mathbf{y}_{train})$ and $\mathcal{L}_{\mathbf{a},\mathbf{h}}(\mathbf{w}, \mathbf{X}_{val}, \mathbf{y}_{val})$ respectively. Additionally, we require (ϵ, δ)-differential privacy w.r.t. model parameters, architectural parameters, and rewards. Loosely speaking, (ϵ, δ)-privacy balances the privacy of user data against the practical usability of the data where a small privacy budget ϵ implies strong privacy guarantees (see App. A.3). We omit DP for now due to readbility.

FEATHERS operates in two stages, the *search stage* and the *evaluation stage*. The search stage alternates between optimizing a neural architecture and optimizing hyperparameters of that search (referred to as phases). The evaluation stage retrains the found architecture, where the HO scheme is applied again to optimize the hyperparameters of the final model. We now describe the HO and NAS phases and provide convergence results in App. A.2.

Algorithm 1: FEATHERS method server side search stage

Data: set of clients $\mathcal{C}$, client weight $v_c \forall c \in \mathcal{C}$, search spaces $\mathcal{H}$ and $\mathcal{A}$, local steps HO e_H, local steps NAS e_N

1 initialize parameters $\mathbf{w}$, $\mathbf{a}$; $\mathbf{r} \leftarrow \mathbf{0}$;
2 $\pi \leftarrow \text{softmax}(\mathbf{r})$;
3 **for** *p in phases* **do**
4 **if** *p == 'ho'* **then**
5 $e \leftarrow p$;
6 sample m configs $\mathbf{h}$ from π;
7 $\mathbf{r}^{(e)} \leftarrow \mathbf{0}$;
8 $\ell_{\mathbf{a},\mathbf{w}}, \ell_{\mathbf{a}',\mathbf{w}'} \leftarrow$ client_step($\mathbf{h}$,$\mathbf{w}$, $\mathbf{a}$, e_H);
9 $\mathbf{r}^{(e)}[h] \leftarrow \sum_{c \in \mathcal{C}} v_c \cdot (\ell^c_{\mathbf{a},\mathbf{w}} - \ell^c_{\mathbf{a}',\mathbf{w}'})$;
10 $\mathbf{r} \leftarrow$ update_rewards($\mathbf{r}$, $\mathbf{r}^{(e)}$);
11 $\pi \leftarrow \text{softmax}(\mathbf{r})$;
12 **if** *p == 'nas'* **then**
13 $\mathbf{h}^* \leftarrow \mathcal{H}[\arg\max_{\mathbf{h}} \mathbf{r}[\mathbf{h}]]$;
14 $\mathbf{w}, \mathbf{a} \leftarrow$ client_step($\mathbf{h}^*$, $\mathbf{w}$, $\mathbf{a}$, e_N);
15 **return**

Hyperparameter Optimization (HO). To identify well-working hyperparameters $\mathbf{h}$ we have to solve $\mathbf{h}^* = \arg\min_{\mathbf{h}} \mathcal{L}_{\mathbf{a}^*,\mathbf{h}}(\mathbf{w}^*; \mathbf{X}_{val}, \mathbf{y}_{val})$. Here, $\mathbf{w}^*$ denote model parameters minimizing the training-loss under architecture $\mathbf{a}^*$ that in turn is a minimizer of the validation-loss under hyperparameters $\mathbf{h} \in \mathcal{H}$ where $\mathcal{H}$ is a discrete set of hyperparameter instantiations. To tackle this problem, a n-armed bandit approach with a strategy similar to ϵ-greedy as shown in Algorithm 1 is employed. After initializing the parameters, architecture and reward estimates **(Line 1-3)**, we randomly sample m hyperparameter configurations from a distribution π over $\mathcal{H}$. All sampled configurations are sent to the clients that perform a few local training steps given the same weights $\mathbf{w}$ and architecture $\mathbf{a}$. This yields an approximation of $\mathbf{a}^*$ and $\mathbf{w}^*$, denoted as $\mathbf{w}'$ and $\mathbf{a}'$ respectively. For each hyperparameter configuration, all clients compute their local validation loss before ($\ell^{(c)}_{\mathbf{a},\mathbf{w}}$) and after ($\ell^{(c)}_{\mathbf{a}',\mathbf{w}'}$) performing local training as shown in Algorithm 2 **(Line 1-11)**. Note that $\mathbf{a}'$ is fixed during evaluation. The reward-signal $r^{(e)}_{\mathbf{h}} = \sum_{c \in \mathcal{C}} v_c \cdot \left(\ell^{(c)}_{\mathbf{a},\mathbf{w}} - \ell^{(c)}_{\mathbf{a}',\mathbf{w}'}\right)$ indicates how well configuration $\mathbf{h}$ performed in HO phase e. After testing each sampled $\mathbf{h}$, in $\mathbf{r}^{(e)}$ each entry corresponds to one hyperparameter configuration in $\mathcal{H}$: For configurations $\mathbf{h}$ sampled in HO round e, $\mathbf{r}^{(e)}$ contains the reward, all other entries are zero. The reward-estimates $\mathbf{r}$ are then updated by $\mathbf{r} = \mathbf{r} + (\mathbf{i} \circ \alpha \cdot (\mathbf{r}^{(e)} - \mathbf{r})) + ((1 - \mathbf{i}) \circ (\alpha \cdot \mathbf{r}) - \mathbf{r})$. Here, $\circ$ is the Hadamard product, $\mathbf{i}$ is a binary vector indicating which hyperparameter configurations were sampled in HO round e and α is a constant factor determining how aggressively the reward estimate is updated. The update rule corrects the rewards for all sampled configurations by the error of the current reward estimate and weights down estimates of all other configurations since the minimizer of the HO problem might change over time due to weight sharing. The hyperparameter configuration maximizing the reward is used in the next NAS phase or training iteration. Besides that, the reward estimates are used to set the distribution $\pi = \text{softmax}(\mathbf{r})$ from which the next configuration candidates are sampled. Also, π is used to determine the number

of communication rounds of the next HO phase, thus controlling the exploration-exploitation trade-off during HO. We compute the number of communication rounds as $\kappa = \text{rnd}(\beta H)$ where β is a constant set by the user and $H = \sum_{\mathbf{h} \in \mathcal{H}} \ln(\pi(\mathbf{h})) \cdot \pi(\mathbf{h})$. If π is a high-entropy distribution (i.e. high H), FEATHERS invests more resources to explore the search space $\mathcal{H}$ while it invests less resources when H is low, i.e. we identified configurations beating most of the others.

Neural Architecture Search. Once the HO-phase yields a hyperparameter configuration $\mathbf{h}$, the architecture is optimized under $\mathbf{h}$ for a certain number of communication rounds as shown in Algorithm 1 **(Line 14-15)**, thereby solving $\mathbf{a}^* = \arg\min_{\mathbf{a}} \mathcal{L}_{\mathbf{a},\mathbf{h}}(\mathbf{w}^*, \mathbf{X}_{val}, \mathbf{y}_{val})$ where $\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathcal{L}_{\mathbf{a},\mathbf{h}}(\mathbf{w}, \mathbf{X}_{train}, \mathbf{y}_{train})$. We follow DARTS (Liu et al., 2019) and apply differentiable NAS on the client side (see App. A.1 for details). To optimize the architecture, first, the architecture is updated by following the gradient $\nabla_{\mathbf{a}} \mathcal{L}_{\mathbf{a},\mathbf{h}}(\hat{\mathbf{w}}, \mathbf{X}_{val}, \mathbf{y}_{val})$ where $\hat{\mathbf{w}} = \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathcal{L}_{\mathbf{a},\mathbf{h}}(\mathbf{w}, \mathbf{X}_{train}, \mathbf{y}_{train})$. As a second step, the model parameters are updated by following the gradient $\nabla_{\mathbf{w}} \mathcal{L}_{\mathbf{a},\mathbf{h}}(\mathbf{w}, \mathbf{X}_{train}, \mathbf{y}_{train})$. As shown in Algorithm 2 **(Line 4-9)**, in each communication round each client c yields new architectural and model parameters $\mathbf{a}'_c$ and $\mathbf{w}'_c$. The differentiable architecture search allows FedAvg to aggregate all clients' model and architecture parameters after each communication round. After the termination of the search, we apply a simple discretization method described in App. A.1.

Differential Privacy (DP). We add independent Gaussian noise to the parameters and rewards sent to the server to achieve (ϵ, δ)-differential privacy of FEATHERS. To achieve differential privacy w.r.t. model- and architectural parameters we follow Abadi et al. (2016) and obtain a new SGD update rule with DP-guarantees: $\theta \leftarrow \alpha_\theta \frac{1}{B} \sum_{i=1}^{B} \mathbf{g}_\theta^{(i)} + \mathcal{N}(0, \sigma_\theta^2 C_\theta^2 \mathbf{I})$. Here, $\theta \in \{\mathbf{w}, \mathbf{a}\}$, i.e. θ refers to the model- or architectural parameters, B denotes the batch size, $\mathcal{N}$ is the normal distribution, σ_θ is a scaling parameter, C_θ is the maximum gradient norm, $\mathbf{I}$ the identity matrix, $\mathbf{g}_\theta^{(i)}$ represents the clipped version of $\nabla_\theta \mathcal{L}_{\mathbf{a},\mathbf{h}}(\mathbf{w}, \mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ and α_θ is the learning rate for parameters θ. The scaling parameter σ_θ inversely depends on ϵ s.t. lower ϵ imply larger noise variance, i.e., stronger privacy guarantees. We apply DP to the HO rewards analogously. We prove privacy guarantees of FEATHERS in App. A.3 and show pseudo-code with DP.

4 Experiments and Results

To empirically demonstrate FEATHER's capabilities, we aim to answer the following questions: **(Q1)** How does FEATHERS joint architecture and hyperparameter search compare to prominent HO- and NAS methods in FL settings at various scales and label skews? **(Q2)** How well does FEATHERS perform if DP is employed to preserve privacy at privacy budget ϵ?

Experimental Protocol. FEATHERS was evaluated against DARTS and FedEx on Fashion-MNIST, CIFAR-10, Tiny-Imagenet, and a real-world fraud detection dataset containing anonymized bank account data from bank customers. The task is to predict the fraud risk (high or low) given customer information[1]. All datasets were partitioned randomly on a set of clients such that each client holds approximately the same number of samples. Since in FL, it is common to have skewed data across clients, we conducted experiments with label skew in the data (referred to as *ls*; see App. B.2). The architecture search space contained CNN/MLP architectures, and the hyperparameter search space contained five real-valued hyperparameters (see App. B.3). The best-performing cell architecture found during the search was discretized with $k = 2$ (see App. A.1) and used to build and train a larger validation network from scratch while employing FEATHER's HO stage with a slightly different hyperparameter search space (see App. B.3). Since we assume a cross-silo setting, we allowed all clients to participate in each communication round. Additionally, we evaluated FEATHERS with and without DP for fraud detection to show that adding DP does not prevent learning a suitable architecture. For further details, refer to App. B.

[1]The dataset is available at `https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud`

Dataset	Fashion-MNIST		CIFAR-10		Tiny-Imagenet	
	w/o ls	w/ ls	w/o ls	w/ ls	w/o ls	w/ ls
DARTS (f, 10 clients)†	0.91 ± 0.02	0.92 ± 0.02	0.91 ± 0.03	0.89 ± 0.04	0.68 ± 0.02	0.67 ± 0.03
DARTS (f, 100 clients)†	0.92 ± 0.03	0.91 ± 0.03	0.91 ± 0.02	0.89 ± 0.03	0.67 ± 0.02	0.67 ± 0.03
FedEx (10 clients)*	0.78 ± 0.03	0.78 ± 0.02	0.51 ± 0.04	0.51 ± 0.03	0.41 ± 0.03	0.40 ± 0.04
FedEx (100 clients)*	0.65 ± 0.03	0.64 ± 0.04	0.46 ± 0.05	0.47 ± 0.04	0.38 ± 0.04	0.38 ± 0.05
FEATHERS (10 clients)	0.93 ± 0.01	0.93 ± 0.03	0.92 ± 0.02	0.89 ± 0.04	0.68 ± 0.02	0.68 ± 0.02
FEATHERS (100 clients)	0.94 ± 0.02	0.93 ± 0.03	0.90 ± 0.03	0.89 ± 0.03	0.68 ± 0.03	0.67 ± 0.03

*Training performed using architecture found by DARTS.
†The same hyperparameter-settings as described in (Liu et al., 2019) were used.

Table 2: **FEATHERS outperforms prominent methods for NAS (DARTS) and HO (FedEx) on various image classification tasks.** The mean accuracy and standard deviation are reported for 5 repeats of each experiment. Colors are interpolated from green to blue (high to low accuracy).

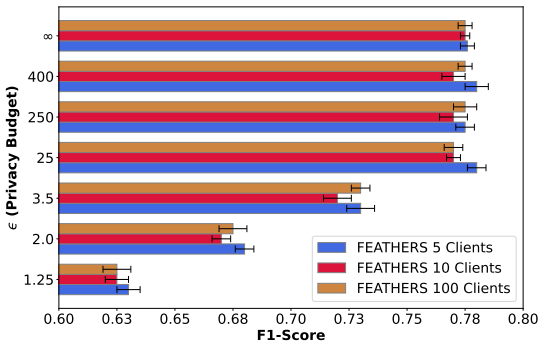

Figure 2: **FEATHERS achieves the same performance with DP as without DP for reasonable ϵ.** The performance decreases for low ϵ (stronger privacy guarantees). We set $\delta = \frac{1}{2 \cdot S}$ where S is the size of the dataset.

Results. Our experiments show that FEATHERS beats the baselines on Fashion-MNIST, CIFAR-10, and Tiny-Imagenet (see Tab. 2). Despite DARTS using hyperparameters that have been tuned manually, our method beats DARTS (94% vs. 92% on Fashion-MNIST, 93% vs. 92% on CIFAR-10, 69% vs. 67% on Tiny-Imagenet) in most distributed learning settings while optimizing for a larger set of parameters, even in label-skew scenarios.

To assess the scalability of FEATHERS, we performed experiments for an increasing number of clients (see Tab. 2). We find that, in contrast to FedEx, the number of participating clients does not seem to influence the performance of FEATHERS negatively and hypothesize that FEATHER's stability is due to more extensive exploration during HO. Each client holds a smaller subset of data for an increasing number of clients since the datasets used have fixed sizes. Thus, the stochastic gradients per client tend to have a higher variance, leading to higher parameter changes across clients. While FEATHERS invests more time in exploring good hyperparameters, FedEx directly applies sampled configurations. Thus, FedEx is more likely to use bad performing hyperparameters, especially with noisy gradients.

Finally, Fig. 2 shows that FEATHERS identifies high-performing architectures and hyperparameters while giving privacy guarantees well-suited for practical purposes (i.e., $\epsilon \geq 25$). As expected, the performance deteriorates for small values of ϵ, i.e., larger noise variances. We thus answer **(Q1)** and **(Q2)** affirmatively and conclude that FEATHERS identifies high-performing architectures and hyperparameters while providing privacy guarantees. See App. B.4 for more results.

5 Broader Impact & Limitations

FEATHERS provides a novel practical way to jointly perform architecture and hyperparameter optimization in sensitive data regimes such as healthcare. Also, we hope that this work serves as a starting point for further HO- and NAS methods in FL with privacy guarantees. The main limitation of FEATHERS is that it requires a discrete hyperparameter search space. Additionally, the HO algorithm is stateless at this stage of development and could be extended to incorporate the current training state (e.g., the architecture). We envision to improve upon that in future work.

6 Conclusion

We introduced FEATHERS, a federated learning method that efficiently optimizes both neural architectures and hyperparameters jointly while providing provable privacy guarantees. Our empirical investigation demonstrates that FEATHERS is more than competitive with prominent NAS- and HO algorithms while optimizing a larger space of hyperparameters.

Acknowledgements. This work was supported by the National High-Performance Computing Project for Computational Engineering Sciences (NHR4CES). Furthermore, this work benefited from the cluster project "The Third Wave of AI". The Eindhoven University of Technology authors received support from their Department of Mathematics and Computer Science and the Eindhoven Artificial Intelligence Systems Institute.

References

Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., and Zhang, L. (2016). Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

Agrawal, S., Sarkar, S., Alazab, M., Maddikunta, P. K. R., Gadekallu, T. R., and Pham, Q.-V. (2021). Genetic cfl: Hyperparameter optimization in clustered federated learning. *Computational Intelligence and Neuroscience*.

Dai, Z., Low, B. K. H., and Jaillet, P. (2021). Differentially private federated bayesian optimization with distributed exploration. *Advances in Neural Information Processing Systems (NeurIPS)*.

Dong, X. and Yang, Y. (2020). Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*.

Dwork, C. (2006). Differential privacy. In Bugliesi, M., Preneel, B., Sassone, V., and Wegener, I., editors, *Automata, Languages and Programming*.

Dwork, C. and Roth, A. (2014). The algorithmic foundations of differential privacy. page 211–407.

Dwork, C., Roth, A., et al. (2014). The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*.

Dwork, C., Rothblum, G. N., and Vadhan, S. (2010). Boosting and differential privacy. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 51–60.

Fredrikson, M., Jha, S., and Ristenpart, T. (2015). Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

Haddadpour, F. and Mahdavi, M. (2019). On the convergence of local descent methods in federated learning. *CoRR*, abs/1910.14425.

He, C., Mushtaq, E., Ding, J., and Avestimehr, S. (2020a). Fednas: Federated deep learning via neural architecture search. *Workshop on Neural Architecture Search and Beyond for Representation Learning (CVPR)*.

He, C., Ye, H., Shen, L., and Zhang, T. (2020b). Milenas: Efficient neural architecture search via mixed-level reformulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Huang, H., Zhang, Z., Shen, Y., Backes, M., Li, Q., and Zhang, Y. (2022). On the privacy risks of cell-based NAS architectures. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. ACM.

Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al. (2021). Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*.

Khodak, M., Tu, R., Li, T., Li, L., Balcan, M.-F. F., Smith, V., and Talwalkar, A. (2021). Federated hyperparameter tuning: Challenges, baselines, and connections to weight-sharing. In *Advances in Neural Information Processing Systems*, volume 34, pages 19184–19197. Curran Associates, Inc.

Koskela, A. and Honkela, A. (2018). Learning rate adaptation for federated and differentially private learning. *arXiv:1809.03832*.

Liu, H., Simonyan, K., and Yang, Y. (2019). Darts: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. (2017). Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*.

McSherry, F. D. (2009). Privacy integrated queries: An extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, page 19–30.

Mostafa, H. (2019). Robust federated learning through representation matching and adaptive hyper-parameters. *arXiv:1912.13075*.

Papernot, N. and Steinke, T. (2022). Hyperparameter tuning with renyi differential privacy.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. (2018). Efficient neural architecture search via parameter sharing. *CoRR*.

Singh, I., Zhou, H., Yang, K., Ding, M., Lin, B., and Xie, P. (2020). Differentially-private federated neural architecture search. *arXiv:2006.10559*.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*.

Ye, J., Maddi, A., Murakonda, S. K., Bindschaedler, V., and Shokri, R. (2022). Enhanced membership inference attacks against machine learning models. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

Zela, A., Klein, A., Falkner, S., and Hutter, F. (2018). Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. *CoRR*, abs/1807.06906.

Zhou, Y., Ram, P., Salonidis, T., Baracaldo, N., Samulowitz, H., and Ludwig, H. (2021). Flora: Single-shot hyper-parameter optimization for federated learning. *arXiv:2112.08524*.

Zhu, H. and Jin, Y. (2021). Real-time federated evolutionary neural architecture search. *IEEE Transactions on Evolutionary Computation*, 26(2):364–378.

Zhu, H., Zhang, H., and Jin, Y. (2021). From federated learning to federated neural architecture search: a survey. *Complex & Intelligent Systems*.

Zoph, B. and Le, Q. (2017). Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*.

Submission Checklist

1. For all authors...

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] In section 4 we have shown that FEATHERS outperforms prominent NAS and HO methods in FL.

   (b) Did you describe the limitations of your work? [Yes] See Section 5.

   (c) Did you discuss any potential negative societal impacts of your work? [No] We don't think that our work presents any notable or specific negative impacts.

   (d) Did you read the ethics review guidelines and ensure that your paper conforms to them? `https://2022.automl.cc/ethics-accessibility/` [Yes]

2. If you ran experiments...

   (a) Did you use the same evaluation protocol for all methods being compared (e.g., same benchmarks, data (sub)sets, available resources)? [Yes] All experiments were run on the same machines, additionally each dataset has used performance metrics from previous publications to allow easy comparability of results

   (b) Did you specify all the necessary details of your evaluation (e.g., data splits, pre-processing, search spaces, hyperparameter tuning)? [Yes] See Sec.4 as well as Appendix C

   (c) Did you repeat your experiments (e.g., across multiple random seeds or splits) to account for the impact of randomness in your methods or data? [Yes] Yes each experiment was repeated 5 times and each time a different random seed was used

   (d) Did you report the uncertainty of your results (e.g., the variance across random seeds or splits)? [Yes] See Table 3 and 2

   (e) Did you report the statistical significance of your results? [Yes] Spread and error metrics are reported. (e.g. Table 3) Estimates of significance between different experiments however are not given, as a sample size of 5 runs per experiment is not sufficient enough to do that

   (f) Did you use tabular or surrogate benchmarks for in-depth evaluations? [No] No, since not all datasets we used are available in benchmarks (e.g. Fraud detection dataset).

   (g) Did you compare performance over time and describe how you selected the maximum duration? [Yes] We report the final performance and final runtime (App. C).

   (h) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See appendix B

   (i) Did you run ablation studies to assess the impact of different components of your approach? [Yes] We analyzed the benefit of an additional HO phase during evaluation, see App. B.4

3. With respect to the code used to obtain your results...

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit versions), random seeds, an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [Yes] Yes a link to the code was provided in the Appendix.

   (b) Did you include a minimal example to replicate results on a small subset of the experiments or on toy data? [Yes] A readme is given in the repository which shows how to setup and run the code.

(c) Did you ensure sufficient code quality and documentation so that someone else can execute and understand your code? [Yes] See readme and Code itself

(d) Did you include the raw results of running your experiments with the given code, data, and instructions? [No] Experiments were conducted over a prolonged time period and not all raw results exsist anymore.

(e) Did you include the code, additional data, and instructions needed to generate the figures and tables in your paper based on the raw results? [Yes] The repository contains code to produce the plots.

4. If you used existing assets (e.g., code, data, models)…

(a) Did you cite the creators of used assets? [Yes] For widely used datasets such as CIFAR-10 and Imagenet, we did not cite; for fraud detection, we provided a link to Kaggle.

(b) Did you discuss whether and how consent was obtained from people whose data you're using/curating if the license requires it? [N/A] public data used

(c) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] No, the fraud detection dataset was anonymized by the providers and thus does not contain personal information of bank customers anymore.

5. If you created/released new assets (e.g., code, data, models)…

(a) Did you mention the license of the new assets (e.g., as part of your code submission)? [N/A]

(b) Did you include the new assets either in the supplemental material or as a URL (to, e.g., GitHub or Hugging Face)? [N/A]

6. If you used crowdsourcing or conducted research with human subjects…

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

7. If you included theoretical results…

(a) Did you state the full set of assumptions of all theoretical results? [Yes] Yes, see Appendix B.

(b) Did you include complete proofs of all theoretical results? [Yes] Yes, see Appendix B.

A FEATHERS Details

Here we give further details on the FEATHERS algorithms.

A.1 Neural Architecture Search

Inspired by Differentiable Architecture Search (DARTS) (Liu et al., 2019) we solve this optimization problem as follows: We define our search space to be a space over *cells*. A cell is a Directed Acyclic Graph (DAG) in which each node is a feature representation and each edge is a *mixed operation*. The feature representation of some node z is computed using all its parent-nodes and the mixed operations defining the edges between z and its parent, i.e. for some node z_j the representation is computed as: $z_j = \sum_{i<j} o^{(z_i,z_j)}(\mathbf{x}^{z_i})$ Here, $o^{(z_i,z_j)}$ is a mixed operation and $\mathbf{x}^{z_i}$ is the feature representation of node z_i. A mixed operation connecting nodes z_1 and z_2 is defined as a weighted sum over a set of operations $\mathcal{O}$:

$$o^{(z_1,z_2)} = \sum_{o \in \mathcal{O}} \frac{\exp(a_o^{(z_1,z_2)})}{\sum_{o' \in \mathcal{O}} \exp(a_{o'}^{(z_1,z_2)})} o(\mathbf{x}) \tag{2}$$

Here, $a_o^{(z_i,z_j)}$ are the architectural parameters to be learned. We use two types of cells: Normal cells and reduction cells. Normal cells keep the dimensions of the input, while reduction cells apply an additional reduction operation.

Discretization. Since we employ a continuous relaxation of the architecture space $\mathcal{A}$, the architecture learned by FEATHERS has to be discretized after training. This is done by selecting the top k operations with the highest architectural weight over all cells. Also, no operation is allowed to connect the same two nodes. The discretized architecture is then retrained in the evaluation stage where only the HO phase from Algorithm 1 is applied.

Algorithm 2: FEATHERS method client side search stage

Data: Network parameters $\mathbf{w}$, architecture $\mathbf{a}$, hyperparameter configurations $\mathbf{H}$, data $\mathbf{X}_{train}$, $\mathbf{X}_{val}$, $\mathbf{y}_{train}$ $\mathbf{y}_{val}$, local steps e

for $\mathbf{h} \in \mathbf{H}$ **do**
  $\mathbf{l}_1, \mathbf{l}_2 \leftarrow [], []$;
  $l_1 \leftarrow \mathcal{L}_{\mathbf{a},\mathbf{h}}(\mathbf{w}, \mathbf{X}_{val}, \mathbf{y}_{val})$;
  for $i \in [e]$ **do**
    $\mathbf{w}^* \leftarrow SGD(\nabla_{\mathbf{w}} \mathcal{L}_{\mathbf{a},\mathbf{w}}(\cdot), \mathbf{w}, \mathbf{h})$;
    $\mathbf{g}_{\mathbf{a}} \leftarrow \text{clip}(\nabla_{\mathbf{a}} \mathcal{L}_{\mathbf{a},\mathbf{w}^*}(\cdot))$;
    $\mathbf{a} \leftarrow SGD(\frac{1}{B}\sum_{i=1}^{B} \mathbf{g}_{\mathbf{a},i}, \mathbf{a}, \mathbf{h})$;
    $\mathbf{g}_{\mathbf{w}} \leftarrow \text{clip}(\nabla_{\mathbf{w}} \mathcal{L}_{\mathbf{a},\mathbf{w}}(\cdot))$;
    $\mathbf{w} \leftarrow SGD(\frac{1}{B}\sum_{i=1}^{B} \mathbf{g}_{\mathbf{w},i}, \mathbf{w}, \mathbf{h})$;
  $l_2 \leftarrow \mathcal{L}_{\mathbf{a},\mathbf{h}}(\mathbf{w}, \mathbf{X}_{val}, \mathbf{y}_{val})$;
  append l_1 to $\mathbf{l}_1$ and l_2 to $\mathbf{l}_2$;
  if p == *ho* **then**
    reset $\mathbf{w}$ and $\mathbf{a}$ to the ones passed;
return $\mathbf{l}_1$, $\mathbf{l}_2$, $\mathbf{w}$, $\mathbf{a}$;

Algorithm 3: FEATHERS Framework Client-side Evaluation stage

Data: Network parameters $\mathbf{w}$, architecture $\mathbf{a}$, hyperparameter configuration $\mathbf{h}$, data $\mathbf{X}_{train}$, $\mathbf{X}_{val}$, $\mathbf{y}_{train}$ $\mathbf{y}_{val}$, local steps e

$l_1 \leftarrow \mathcal{L}_{\mathbf{a},\mathbf{h}}(\mathbf{w}, \mathbf{X}_{val}, \mathbf{y}_{val})$;
for $i \in [e]$ **do**
  $\mathbf{w} \leftarrow SGD(\nabla_{\mathbf{w}} \mathcal{L}_{\mathbf{a},\mathbf{h}}(\cdot), \mathbf{w}, \mathbf{h})$;
$l_2 \leftarrow \mathcal{L}_{\mathbf{a},\mathbf{h}}(\mathbf{w}, \mathbf{X}_{val}, \mathbf{y}_{val})$;
return l_1, l_2, $\mathbf{w}$, $\mathbf{a}$;

A.2 Convergence Analysis

We that FEATHERS' convergence properties in distributed settings coincide with the convergence properties of DARTS in centralized settings with high probability, only scaled by a controllable factor arising from using FedAvg. For simplicity, we do not consider DP in our analysis.

Theorem 1. *Given a joint distribution $p(X_1, \ldots, X_n, y)$ over random variables $X_1, \ldots, X_n, y$ from which each client $c \in \mathcal{C}$ of a set of clients $\mathcal{C}$ samples a dataset $\langle \mathbf{X}^{(c)}, \mathbf{y}^{(c)} \rangle \sim p$, FEATHERS enjoys the same convergence properties as DARTS in a centralized setting if applied on a dataset $\langle \mathbf{X}, \mathbf{y} \rangle$ where $\mathbf{X} = \bigcup_{c \in \mathcal{C}} \mathbf{X}^{(c)}$ and $\mathbf{y} = \bigcup_{c \in \mathcal{C}} \mathbf{y}^{(c)}$.*

Proof. We treat the HO-phase of FEATHERS as an oracle and assume that it returns optimal hyperparameters $\mathbf{h}^*$. Once $\mathbf{h}^*$ was obtained, it is fixed for a certain number of communication rounds κ. In each communication round i epochs of DARTS are performed locally on each of the C clients. Since we employ FedAvg to average model parameters after i local epochs, we exploit that FedAvg converges with rate $\mathcal{O}(\frac{1}{\sqrt{C\kappa}})$ under the assumption that the loss is L-smooth and the μ-Polyak-Łojasiewicz (PL) assumption holds (Haddadpour and Mahdavi, 2019). Since FedAvg converges and parameter-updates are only propagated during NAS-phases, it follows that FEATHERS enjoys the same convergence properties as DARTS in each NAS-phase scaled by the convergence of FedAvg $\mathcal{O}(\frac{1}{\sqrt{C\kappa}})$. □

Since the above proof assumes that our method selects optimal hyperparameters $\mathbf{h}^*$ for each NAS-phase, we will now show that the HO-phase converges with high probability in non-stationary bandit-environments.

Theorem 2. *Given a fixed hyperparameter-space $\mathcal{H}$ and noisy, non-stationary rewards $r_{\mathbf{h}}^{(j)} \sim \mathcal{N}(\mu_{\mathbf{h}}^{(j)}, \sigma_{\mathbf{h}})$ where $\mu_{\mathbf{h}}^{(j)}$ is the expected value of the reward at iteration j, $\sigma_{\mathbf{h}}$ its standard deviation and $\mathbf{h} \in \mathcal{H}$, the HO-strategy of FEATHERS is at most off by $\alpha \cdot 3\sigma_{\mathbf{h}}$ for learning rate α with probability 0.997 once $\mathbf{h} \in \mathcal{H}$ is sampled.*

Proof. Our proof is inspired by convergence results for ϵ-greedy strategies as stated in (Sutton and Barto, 2018). We assume that $|\mu_{\mathbf{h}}^{(j+1)} - \mu_{\mathbf{h}}^{(j)}| \leq \delta$ for finite $\delta \in \mathbb{R}$ in all iterations and $0 < \alpha < 1$ in the update rule. Since the softmax-function cannot evaluate to a point-mass, we can make a strict positivity assumption of the distribution over hyperparameters, i.e. $\pi_i[\mathbf{h}] > 0$ for all $\mathbf{h} \in \mathcal{H}$. Thus, with j approaching infinity, each $\mathbf{h} \in \mathcal{H}$ will be sampled infinitely many times. At an iteration j, in the most extreme case, a certain $\mathbf{h} \in \mathcal{H}$ has not been sampled yet. Assume it gets sampled in iteration j. Since $r_{\mathbf{h}}^{(j)} \sim \mathcal{N}(\mu_{\mathbf{h}}^{(j)}, \sigma_{\mathbf{h}})$ and the current estimate reward-estimate $\mathbf{r}_{\mathbf{h}} = 0$, the update rule reads: $\mathbf{r}_{\mathbf{h}} = \alpha \cdot r_{\mathbf{h}}^{(j)}$. Since we assume all rewards being Gaussian distributed, the probability of obtaining a reward $r_{\mathbf{h}}^{(j)}$ in the range of $3\sigma_{\mathbf{h}}$ is 0.997. Since $0 < \alpha < 1$ holds, our estimate is at most $\pm\alpha \cdot 3\sigma_{\mathbf{h}}$ of w.r.t. $\mu_{\mathbf{h}}^{(j)}$ in 99.7% of the cases. □

As the above only considers the case in which our algorithm terminates after some $\mathbf{h} \in \mathcal{H}$ is sampled, we also have to consider the following case: Assume $\mathbf{h}$ is sampled at iteration j and a reward-estimate is obtained. After that, $\mathbf{h}$ is not sampled for k subsequent iterations. The following theorem gives bounds for how much off our estimate will be in this case.

Theorem 3. *Under the assumptions of Theorem 2, the reward estimate $r_{\mathbf{h}}^{(j+k)}$ will be at most off by $\alpha^k r_{\mathbf{h}}^{(j)} - (k\delta + \mu_{\mathbf{h}}^{(j)})$ assuming that $\mathbf{h}$ is sampled at iteration j and not sampled for k subsequent iterations.*

Proof. By assumptions from Theorem 2, the mean will be shifted by at most $k\delta$ after k steps. Since the update rule for $r_{\mathbf{h}}$ is defined as $r_{\mathbf{h}} = \alpha r_{\mathbf{h}}$, the reward estimate after k iterations in which $\mathbf{h}$ is not sampled is $\alpha^k r_{\mathbf{h}}^{(j)}$. It follows that, k iterations after $\mathbf{h}$ was sampled, the reward estimate is off by at most $\alpha^k r_{\mathbf{h}}^{(j)} - (k\delta + \mu_{\mathbf{h}}^{(j)})$. □

It turns out that the above bound can be controlled by setting $\alpha \leq (1 + \frac{k\delta}{\mu^{(j)}})^{\frac{1}{k}}$ assuming we have access to $\mu^{(j)}$ (see below). In the case $\mu^{(j+1)} - \mu^{(j)} = \delta$, this relation guarantees that our reward estimate of some $\mathbf{h}$ is still optimal if $\mathbf{h}$ was not sampled for k HO-rounds. Since we can assume that the loss decreases between HO-rounds, i.e. $\mu^{(j+1)} - \mu^{(j)} < 0$, the assumption $0 < \alpha < 1$ used in the above theorems is not violated. Using Theorem 2, we can assume that we have an estimate of $\mu^{(j)}$ fulfilling at least $\mu^{(j)} \pm \alpha \cdot 3\sigma_{\mathbf{h}}$ with high probability for some $\mathbf{h}$ sampled the first time in round j. Hence, the errors of reward estimates can be controlled within the reasonable bound given by Theorem 3 in subsequent rounds.

Selection of α. Based on the assumptions of Theorem 2 and 3, we have true mean-values of rewards $\mu^{(j)}$ and $\mu^{(j+1)}$ for round j and $j+1$ respectively. Also, we assume that $|\mu^{(j+1)} - \mu^{(j)}| \leq \delta$ for all j. Let's assume we have access to $\mu^{(j)}$ at round j for some $\mathbf{h}$ and that $\mathbf{h}$ is not sampled for k subsequent rounds. Then our estimate of $\mu^{(j+k)}$ would be $\alpha^k \mu^{(j)}$ after k rounds. To obtain the correct estimate of $\mu^{(j+k)}$, the following must hold:

$$\alpha^k \cdot \mu^{(j)} = \mu^{(j)} + k \cdot (\mu^{(j+1)-\mu^{(j)}}) \tag{3}$$

We can find the optimal α using the following derivation:

$$\alpha^k \cdot \mu^{(j)} = \mu^{(j)} + k \cdot (\mu^{(j+1)-\mu^{(j)}}) \tag{4}$$

$$\alpha^k = 1 + \frac{k \cdot (\mu^{(j+1)} - \mu^{(j)})}{\mu^{(j)}} \tag{5}$$

$$\leq 1 + \frac{k\delta}{\mu^{(j)}} \tag{6}$$

It follows that

$$\alpha = \left(1 + \frac{k\delta}{\mu^{(j)}}\right)^{\frac{1}{k}} \tag{7}$$

Note that in case $\mu^{(j+1)} - \mu^{(j)} = \delta$ by applying the above equation we obtain the optimal α.

A.3 Differential Privacy

Although in FL no data is exchanged between server and clients, the parameters sent to the server still leak private information (Fredrikson et al., 2015; Ye et al., 2022; Huang et al., 2022; Papernot and Steinke, 2022). Differential privacy (DP) is an effective way to provably protect private information encoded e.g. parameters during Stochastic Gradient Descent (SGD) (Abadi et al., 2016). We adapt

this notion and apply DP to all messages being sent during training that might carry private information.

The Algorithm. Let us start off by revisiting the definition of DP which was introduced in Dwork (2006):

Definition 1 ((ϵ, δ)-Differential Privacy). *For any two datasets D, D' that differ in exactly one record a mechanism M is called ϵ-δ-differential private if $\forall x : Pr[M(D) = x] \leq \exp(\epsilon)Pr[M(D') = x] + \delta$ holds where $Pr[M(D) = x]$ is the probability of mechanism M outputting x if executed on D.*

In our case M is the learning procedure, i.e. SGD. Making SGD differential private can be achieved by clipping gradients and adding Gaussian noise to the gradient of each sample w.r.t. the parameters, resulting in an algorithm called DP-SGD (Abadi et al., 2016). Since we update both, model- and architectural parameters using SGD we simply can follow Abadi et al. (2016) and obtain the following SGD update rules with DP-guarantees:

$$\theta \leftarrow \alpha_\theta \frac{1}{B} \sum_{i=1}^{B} \mathbf{g}_\theta^{(i)} + \mathcal{N}(0, \sigma_\theta^2 C_\theta^2 \mathbf{I}) \tag{8}$$

In the above equation we use $\theta \in \{\mathbf{w}, \mathbf{a}\}$ to either refer to the model- or architectural parameters. B denotes the batch-size, $\mathcal{N}$ is the normal distribution, σ_θ is a scaling-parameter, C_θ is the maximum gradient norm, $\mathbf{I}$ the identity matrix, $\mathbf{g}_\theta^{(i)}$ represents the clipped version of $\nabla_\theta \mathcal{L}_{\mathbf{a},\mathbf{h}}(\mathbf{w}, \mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ and α_θ is the learning rate for parameters θ. We use DP-SGD for learning both, the model parameters and the architecture. ϵ inversely depends on noise-parameters σ, hence for high ϵ-values DP-SGD achieves approximately SGD-convergence while losing privacy-guarantees. For low ϵ we obtain strong privacy guarantees while giving up convergence-guarantees Abadi et al. (2016). It should be noted that FedAvg averages the parameters that have been computed by the clients. Since DP is closed under arbitrary post-processing, averaging does not break DP (Dwork et al., 2014). Algorithm 4 performs the exact same computations as Algorithm 2 except that it adds independent noise N to the clipped gradients w.r.t. model- and architecture parameters of each batch **(Line 3-8)**.

The results shown in (Papernot and Steinke, 2022) suggest that hyperparameters possibly leak private information via evaluation metrics. Since the loss of a model can be seen as such a metric, we add Gaussian noise to the losses computed before and after one optimization step on client-side to achieve (ϵ, δ)-differential privacy w.r.t. the hyperparameters. To that end, independent Gaussian noise is added to the losses computed on client side which are used to compute the rewards for each hyperparameter configuration **(Line 1, 9)**. The noise is drawn from a Gaussian distribution with zero mean and variance depending on the privacy budget ϵ (lower ϵ means higher variance).

Algorithm 5 shows the DP variant of the evaluation stage and performs the same computation as Algorithm 3. Again, the only difference is that the DP variant adds independent Gaussian noise N with zero mean and variance depending on ϵ to the clipped gradients of a batch and independent Gaussian noise to the losses computed on client side **(Line 1-6)**.

As for privacy-related applications it is crucial to guarantee a certain level of privacy, we proceed with deriving privacy guarantees for FEATHERS.

Privacy Guarantees of FEATHERS. It is worth noting that the architecture and hyperparameters are optimized w.r.t. validation data while model parameters are optimized w.r.t. training data. Hence we have to provide guarantees for both datasets. Further, both datasets are distributed over $|\mathcal{C}|$ clients. For our analysis we decompose FEATHERS into three differentially private subroutines: Let M_1 be HO phase, M_2 DP-SGD adapting architecture parameters and M_3 DP-SGD adapting model weights. We now state established theorems which our subsequent derivation is based on. First note that FEATHERS is executed multiple times on the same data. Hence the Basic Composition theorem applies in our case which reads:

Theorem 4 (Basic Composition (Dwork et al., 2010)). *For mechanisms $M_1, \ldots, M_k$ each being (ϵ, δ)-differentially private and data x the mechanism $M(x) = (M_1(x), \ldots, M_k(x))$ enjoys $(k\epsilon, k\delta)$-differential privacy.*

Additionally, the HO (M_1) and NAS (M_2) phases are called subsequently, i.e. M_2 makes use of the result of M_1 which allows us to apply the Adaptive Composition theorem:

Theorem 5 (Adaptive Composition (Dwork et al., 2010)). *For any $\hat{\epsilon} > 0, \hat{\delta} \in [0, 1]$, $\delta' \in (0, 1]$, data x and k $(\hat{\epsilon}, \hat{\delta})$-differentially private mechanisms $M_1, \ldots, M_k$ where each M_j takes x and the result of M_{j-1} as input, the mechanism $M_{1\ldots k}(x) = M_k(x, M_{1\ldots k-1}(x))$ enjoys (ϵ, δ)-differential privacy where $\delta = k\hat{\delta} + \delta'$ and $\epsilon = k\hat{\epsilon}(e^{\hat{\epsilon}} - 1) + \hat{\epsilon}\sqrt{2k\ log(1/\delta')}$.*

As we assume that each client holds a distinct dataset, the Parallel Composition theorem applies to FEATHERS as well:

Theorem 6 (Parallel Composition (McSherry, 2009)). *For mechanisms $M_1, \ldots, M_k$ each being (ϵ, δ)-differentially private and disjoint datasets $x_1, \ldots, x_k$ the mechanism $M(x) = (M_1(x_1), \ldots, M_k(x_k))$ enjoys (ϵ, δ)-differential privacy.*

FEATHERS aggregates model- and architecture parameters via FedAvg and aggregates rewards from the HO phase. The Post-Processing theorem guarantees that these post-processing operations don't leak private information after aggregation:

Theorem 7 (Post-Processing (Dwork and Roth, 2014)). *For a mechanism M being (ϵ, δ)-differentially private and any function $f : range(M) \longrightarrow R$ mapping outputs of M to some set R, $f(M(x))$ is still (ϵ, δ)-differential privacy for data x.*

Before we start our analysis, the following remark is useful for our analysis as M_2 and M_3 are DP-SGD instances.

Remark 1 (Privacy of DP-SGD (Abadi et al., 2016)). *Assuming each step of DP-SGD is (ϵ, δ)-differential private, DP-SGD provides $(\mathcal{O}(q\epsilon\sqrt{E}), \delta)$-differential privacy where E is the number of training epochs and q the probability of each batch.*

We now can state the privacy guarantee of FEATHERS.

Theorem 8 (Privacy of FEATHERS). *Assume M_1, M_2 to be (ϵ_H, δ_H) and M_3 to be (ϵ_D, δ_D)-differentially private respectively. Further assume $\mathbf{X}_{val}^{(c)} \cap \mathbf{X}_{train}^{(c)} = \emptyset$ for all $c \in \mathcal{C}$ and that client's datasets are disjoint. Then, FEATHERS is $(K\epsilon, K\delta)$-differential private where $\epsilon = \max\left\{\mathcal{O}((q+1)\epsilon_H\sqrt{E})\left((e^{\epsilon_H} - 1) + \sqrt{4\ log\frac{1}{\delta'}}\right), Kq\epsilon_D\sqrt{D}\right\}$ and $\delta = \max\{2\delta_H + \delta', \delta_D\}$ where K is the number of communication rounds, q the probability of some batch being used by DP-SGD and $\delta' \in (0, 1]$.*

Proof. Performing the HO phase once and doing E differential private SGD step w.r.t. the architecture on a client c can be represented by $M(\mathbf{X}_{val}^{(c)}) = M_2(\mathbf{X}_{val}^{(c)}, M_1(\mathbf{X}_{val}^{(c)}))$ using M_1 and M_2. Due to Remark 1 and Theorem 5 we get the following upper bound for ϵ_M:

$$\begin{aligned}
&\mathcal{O}((q+1)\epsilon_H\sqrt{E})(e^{\epsilon_H} - 1) + \mathcal{O}((q+1)\epsilon_H\sqrt{E})\sqrt{4\log\frac{1}{\delta'}} \\
=&\mathcal{O}((q+1)\epsilon_H\sqrt{E})\left((e^{\epsilon_H} - 1) + \sqrt{4\log\frac{1}{\delta'}}\right)
\end{aligned}$$

Note that due to Theorems 7 and 7 the aggregation performed during the HO phase to identify the best hyperparamter configuration does not affect privacy. Due to Theorem 5 $\delta_M = 2\delta_H + \delta'$.

M_3 performs DP-SGD on $\mathbf{X}^{(c)}_{train}$, thus enjoying $(\mathcal{O}(q\epsilon_D\sqrt{E}), \delta_D)$-differential privacy with E local training epochs. Note that M_3 uses the results from M. Since $\mathbf{X}^{(c)}_{train} \cap \mathbf{X}^{(c)}_{val} = \emptyset$, Theorem 6 applies, hence the privacy properties do not change for M_3 w.r.t. the training data. Due to Theorem 6 and 7 ensure that privacy guarantees are retained if M_3 is performed on all clients in parallel and gradients are aggregated. It follows that we can *guarantee* (ϵ, δ)-differential privacy for one step of FEATHERS where $\epsilon = \max\left\{\mathcal{O}((q+1)\epsilon_H\sqrt{E})\left((e^{\epsilon_H}-1) + \sqrt{4\log\frac{1}{\delta'}}\right), Kq\epsilon_D\sqrt{D}\right\}$ and $\delta = \max\{2\delta_H + \delta', \delta_D\}$.

For K communication rounds, Theorem 4 implies that FEATHERS is $(K\epsilon, K\delta)$-differential private. □

Note that if clients can be trusted one can save $\sqrt{E}$ of privacy budget by adding noise only in the last epoch of local training to the gradients w.r.t. architecture- and model parameters, i.e. before sending model information to the server.

Algorithm 4: FEATHERS Framework Client-side Search stage with DP

Data: Parameters $\mathbf{w}$ and architecture $\mathbf{a}$, hyperparameter configurations $\mathbf{H}$, data $\mathbf{X}_{train}$, $\mathbf{X}_{val}$, $\mathbf{y}_{train}$ $\mathbf{y}_{val}$, local steps e

for $\mathbf{h} \in \mathbf{H}$ **do**
  $\mathbf{l}_1, \mathbf{l}_2 \leftarrow [], []$;
  $l_1 \leftarrow \mathcal{L}_{\mathbf{a},\mathbf{h}}(\mathbf{w}, \mathbf{X}_{val}, \mathbf{y}_{val})$;
  for $i \in [e]$ **do**
    $\mathbf{w}^* \leftarrow SGD(\nabla_{\mathbf{w}}\mathcal{L}_{\mathbf{a},\mathbf{w}}(\cdot), \mathbf{w}, \mathbf{h})$;
    $\mathbf{g}_\mathbf{a} \leftarrow \text{clip}(\nabla_\mathbf{a}\mathcal{L}_{\mathbf{a},\mathbf{w}^*}(\cdot))$;
    if $i == e$ **then**
      $\mathbf{g}_\mathbf{a} \leftarrow \mathbf{g}_\mathbf{a} + \mathbf{N}_\mathbf{a}$
    $\mathbf{a} \leftarrow SGD(\frac{1}{B}\sum_{i=1}^{B}\mathbf{g}_{\mathbf{a},i}, \mathbf{a}, \mathbf{h})$;
    $\mathbf{g}_\mathbf{w} \leftarrow \text{clip}(\nabla_\mathbf{w}\mathcal{L}_{\mathbf{a},\mathbf{w}}(\cdot))$;
    if $i == e$ **then**
      $\mathbf{g}_\mathbf{w} \leftarrow \mathbf{g}_\mathbf{w} + \mathbf{N}_\mathbf{w}$
    $\mathbf{w} \leftarrow SGD(\frac{1}{B}\sum_{i=1}^{B}\mathbf{g}_{\mathbf{w},i}, \mathbf{w}, \mathbf{h})$;
  $l_2 \leftarrow \mathcal{L}_{\mathbf{a},\mathbf{h}}(\mathbf{w}, \mathbf{X}_{val}, \mathbf{y}_{val})$;
  append l_1 to $\mathbf{l}_1$ and l_2 to $\mathbf{l}_2$;
  if p == ho **then**
    reset $\mathbf{w}$ and $\mathbf{a}$ to the ones passed;
$\mathbf{l}_1, \mathbf{l}_2 \leftarrow \mathbf{l}_1 + \mathbf{N}_1, \mathbf{l}_2 + \mathbf{N}_2$;
return $\mathbf{l}_1$, $\mathbf{l}_2$, $\mathbf{w}$, $\mathbf{a}$;

Algorithm 5: FEATHERS Framework Client-side Evaluation stage with DP

Data: Parameters $\mathbf{w}$ and architecture $\mathbf{a}$, hyperparameter configuration $\mathbf{h}$, data $\mathbf{X}_{train}$, $\mathbf{X}_{val}$, $\mathbf{y}_{train}$ $\mathbf{y}_{val}$, local steps e

$l_1 \leftarrow \mathcal{L}_{\mathbf{a},\mathbf{w}}(\mathbf{w}, \mathbf{X}_{val}, \mathbf{y}_{val}) + N_{l_1}$;

for $i \in [e]$ **do**

  $\mathbf{g}_{\mathbf{w}} \leftarrow \text{clip}(\nabla_{\mathbf{w}} \mathcal{L}_{\mathbf{a},\mathbf{w}}(\cdot))$;

  $\mathbf{w} \leftarrow SGD(\frac{1}{B} \sum_{i=1}^{B} \mathbf{g}_{\mathbf{w},i} + N, \mathbf{w}, \mathbf{h})$;

$l_2 \leftarrow \mathcal{L}_{\mathbf{a},\mathbf{w}}(\mathbf{w}, \mathbf{X}_{val}, \mathbf{y}_{val}) + N_{l_2}$;

return l_1, l_2, $\mathbf{w}$, $\mathbf{a}$;

B Training Details

In the following, we provide training details of our experiments. We implemented FEATHERS in Python using the flwr framework for federated learning. We provide our code at `https://github.com/ml-research/FEATHERS`.

B.1 General Setup

We trained the supernet in the search stage for 200/500 communication rounds on Fashion-MNIST and CIFAR-10 respectively. The number of communication rounds per HO phase were computed as described in Section 3, NAS was performed for 15 rounds after each HO phase. We set $\beta = 4$ and defined a uniform distribution over the 120 hyperparameter instantiations we use as a search space. We set $\alpha = 0.65$ in all experiments. In each communication round, all selected clients train for 5 epochs. Data was shuffled before distribution on clients using a fixed seed. We used gradient clipping with a value of 5 and chose a batch size of 64. We chose the architecture leading to the highest accuracy score obtained in during the search stage. This architecture was then used to build and train an evaluation network as described in Section 3. Training took place for 500 communication rounds where selected clients perform 5 epochs of training in each communication round. We set $\beta = 4$ and the number of rounds training under a certain hyperparameter instantiation $\mathbf{h}$ to 15 with 5 epochs of local training. We set gradient clipping to 5 and chose a batch size of 96 for Fashion-MNIST/CIFAR-10 and 128 for Tiny-Imagenet to be comparable to DARTS.

B.2 Label Skew

In FL, commonly, data is not equally distributed across clients. One case often encountered in the real world is that labels are unequally distributed across clients. Therefore, we simulated this scenario in our experiments and distributed samples s.t. one client holds $p\%$ of a label l while the remaining $100 - p\%$ of the samples with label l are split equally across clients. In our experiments, we set $p = 66\%$, i.e., one client holds 66% of the samples of a certain label l while the remaining 33% are split equally across all other clients. The label was chosen at random.

B.3 Search Space Image Classification

Search Stage. Our search space for image classification tasks is divided into an architecture search space $\mathcal{A}$ and a non-architectural hyperparameter-search space $\mathcal{H}$. $\mathcal{A}$ was defined as the space of cells consisting of 7 nodes, connected by a mixed operation consisting of the following primitives: Separable/dilated separable convolutions of size 3×3 and 5×5, max- and average pooling of size 3×3, an identity operation and a *zero*-operation. We used a stride of one and padding. Convolutional operations are defined using the ReLU-Conv-BN order and we apply separable convolutions twice. We stack 8 cells s.t. the input of each cell is the output of its last two predecessors and every third cell is a reduction cell, the rest are normal cells. The output of each cell is defined as the

depth-wise concatenation of the representations of its nodes. In case of fraud detection we are using a search space over MLPs since we deal with tabular data. Here, operations are defined as small MLPs which map the input to a lower or higher dimension and which use either Tanh, Sigmoid or ReLU as activation functions. $\mathcal{H}$ consists of candidates for the learning rate used for model- and architecture parameter-updates sampled from a log-uniform distribution from $10^{\exp(\mathcal{U}(-4,0))}$ and $10^{\exp(\mathcal{U}(-5,-1))}$ respectively. Further we included candidates for weight decay used in updates of both parameter-types sampled from $10^{\exp(\mathcal{U}(-5,-1))}$ and we included candidates for the momentum used in SGD-updates of model parameters with momentum candidates sampled from $\mathcal{U}(0.5, 1)$. We used 120 i.i.d. samples from these distributions.

Evaluation Stage. Here we tuned the learning rate, weight decay, momentum and path-dropout (counteracting over-fitting). On Fashion-MNIST and CIFAR-10 we changed allowed values for dropout to values between 0 and 0.5. In the evaluation on Tiny-Imagenet we sampled the learning rates from $10^{\exp(\mathcal{U}(-6,-1))}$, weight-decay from $10^{\exp(\mathcal{U}(-5,-2))}$, momentum and dropout remained the same as for CIFAR-10 and Fashion-MNIST. We sampled 240 candidates instead of 120. For MLPs we replaced path-dropout by regular dropout and sampled 120 i.i.d. samples as above, for path-dropout we sampled from $\mathcal{U}(0, 0.3)$.

B.4 Results

Tab. 3 shows a full version of Tab. 2. We find that FEATHERS is on par with or outperforms the baselines on all tasks. Fig. 3 compares the result of running the evaluation stage of FEATHERS with and without the additional HO phase. For that, we identified an architecture using the search stage of FEATHERS on CIFAR-10 using the NAS-Bench-201 search space. Then, we queried NAS-Bench-201 to obtain the performance of the given architecture when no additional HO is performed. Also, we performed the evaluation stage of FEATHERS, i.e., with additional HO. It can be seen that applying HO significantly improves the final model performance. Also, with an increasing number of clients, the effect of the additional HO increases as well. We suspect that this is due to the following reason: When distributing data across clients, the local datasets' size decreases as we do not allow the same sample to reside on multiple clients. This can lead to higher variances in the gradients during optimization, thus making optimization more unstable and prone to end up with bad parameters. The HO phase during evaluation explores the hyperparameter space periodically and aims to choose the configuration yielding the best improvement *on average across all clients*. This might point FEATHERS towards hyperparameters that "counteract" the increasing variances of gradients (e.g., lower learning rates), thus stabilizing learning and fostering SGD to end up at better solutions. Fig. 4 shows that FEATHERS indeed tends to choose more "cautious" hyperparameters than humans usually use (e.g., low learning rates and lower momentum).

B.5 Runtimes

In terms of runtime, FEATHERS ($\sim$ 2.5 GPU-days) does not add significant overhead compared to DARTS ($\sim$ 2 GPU-days). The additional HO-phase during the search stage adds an overhead of approximately 0.1-0.8 GPU-days, depending on the number of instantiations tested in each HO-round. In contrast, FedEx' runtime ($\sim$ 1 GPU-day) is much lower compared to DARTS and FEATHERS since FedEx does not perform NAS and that it performs less exploration than our method. See Tab. 4 for a detailed listing of runtimes w.r.t. datasets and number of clients.

FEATHERS dynamically adjusts hyperparameters. Figure 4 shows the hyperparameters selected by FEATHERS over time for three runs on CIFAR-10. We observe that our method chooses more "cautious" hyperparameters than engineers usually do. For example, in DARTS it is common to start with a learning rate of 0.025, FEATHERS however chooses much lower learning rates most of the time. Presumably this is due to the properties of our HO-algorithm: In the first HO-round it samples and tests a small subset of instantiations from $\mathcal{H}$ before greedily selecting the one leading to the

Table 3: **Achieved accuracies of FEATHERS and baselines.** DARTS, FedEx, and FEATHERS were compared in different FL settings as described in Section 4. Each experiment was performed 5 times, and the mean accuracy and standard deviation were reported. Colors are interpolated from green to blue (high accuracy to low accuracy).

Dataset	Fashion-MNIST		CIFAR-10		Tiny-Imagenet	
	w/o ls	w/ ls	w/o ls	w/ ls	w/o ls	w/ ls
DARTS (f, 5 clients)†	0.92 ± 0.02	0.93 ± 0.01	0.92 ± 0.02	0.90 ± 0.02	0.67 ± 0.02	0.67 ± 0.02
DARTS (f, 10 clients)†	0.91 ± 0.02	0.92 ± 0.02	0.91 ± 0.03	0.89 ± 0.04	0.68 ± 0.02	0.67 ± 0.03
DARTS (f, 100 clients)†	0.92 ± 0.03	0.91 ± 0.03	0.91 ± 0.02	0.89 ± 0.03	0.67 ± 0.02	0.67 ± 0.03
FedEx (5 clients)*	0.82 ± 0.01	0.81 ± 0.01	0.53 ± 0.02	0.54 ± 0.03	0.43 ± 0.04	0.41 ± 0.04
FedEx (10 clients)*	0.78 ± 0.03	0.78 ± 0.02	0.51 ± 0.04	0.51 ± 0.03	0.41 ± 0.03	0.40 ± 0.04
FedEx (100 clients)*	0.65 ± 0.03	0.64 ± 0.04	0.46 ± 0.05	0.47 ± 0.04	0.38 ± 0.04	0.38 ± 0.05
FEATHERS (5 clients)	0.94 ± 0.01	0.93 ± 0.02	0.93 ± 0.03	0.91 ± 0.03	0.69 ± 0.02	0.69 ± 0.03
FEATHERS (10 clients)	0.93 ± 0.01	0.93 ± 0.03	0.92 ± 0.02	0.89 ± 0.04	0.68 ± 0.02	0.68 ± 0.02
FEATHERS (100 clients)	0.94 ± 0.02	0.93 ± 0.03	0.90 ± 0.03	0.89 ± 0.03	0.68 ± 0.03	0.67 ± 0.03

*Training performed using architecture found by DARTS.
†The same hyperparameter-settings as described in (Liu et al., 2019) were used.

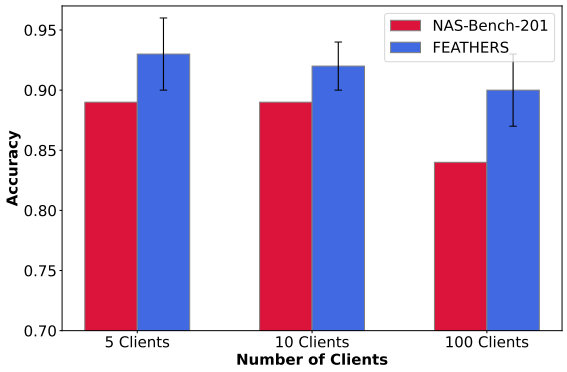

Figure 3: **FEATHERS beats NAS-Bench-201**. FEATHERS' additional HO phase improves the performance of a fixed architecture. Note that NAS-Bench-201 only provides averaged results in centralized settings.

highest decrease in validation loss. In this concrete example, this choice might lead FEATHERS to choose a lower learning rate than 0.025 because there was no better sample. In subsequent HO-rounds the goal is to learn a distribution over instantiations maximizing the reward in the long run. As SGD never truly converges due to its inherent stochasticity, a smaller learning rate is ultimately beneficial in the later stages of training in order to avoid heavily perturbing away from a minimum (i.e. too large learning rates will "overshoot").

Consequently, FEATHER's "cautious" instantiations entail more stable convergence. In that sense, FEATHERS mimics an annealing mechanism in later training stages, which find frequent use in Deep Learning problems. To assess the effect of dynamic hyperparameter adjustments, we compare FEATHERS with NAS-Bench-201 (Dong and Yang, 2020). This benchmark provides a database which allows to query the performance of architectures trained under fixed and manually tuned hyperparameters. The architectures in NAS-Bench-201 were chosen such that they cover widely used architecture search space, including ours. We can easily assess whether our additional HO mechanism helps improving model performance by comparing to NAS-Bench-201: We first

Table 4: **GPU days comparison**. FEATHERS' runtime is approximately 0.4 GPU-days higher than the runtime of DARTS. FedEx has lower runtimes compared to both, FEATHERS and DARTS, mainly because it does not optimize the architecture and performs less exploration than FEATHERS. All runtimes in GPU-days.

	Fashion-MNIST	CIFAR-10	Tiny Imagenet	Fraud Detection
FedEx (5 Clients)	0.8	0.9	1.3	0.09
FedEx (10 Clients)	0.8	0.8	1.3	0.07
FedEx (100 Clients)	0.6	0.7	1.1	0.05
DARTS (5 Clients)	1.8	2.1	3.1	0.4
DARTS (10 Clients)	1.7	2.0	2.9	0.3
DARTS (100 Clients)	1.5	1.8	2.7	0.2
FEATHERS (5 Clients)	2.2	2.5	3.6	0.6
FEATHERS (10 Clients)	2.1	2.4	3.5	0.6
FEATHERS (100 Clients)	1.9	2.1	3.1	0.35

run the search stage of FEATHERS to optimize the architecture for 5/10/100 clients. Note that the architecture found can vary for a different number of clients. Then, we train the architecture found during the search stage using FEATHER's validation stage (i.e. with adjustments of hyperparameters) and compare the accuracy of the same architecture reported in NAS-Bench-201 (i.e. trained with fixed hyperparameters) on CIFAR-10. Figure 3 demonstrates that our dynamic adjustment helps improving model performance. This observation further supports our claim that our method adjusts hyperparameters appropriately over time.

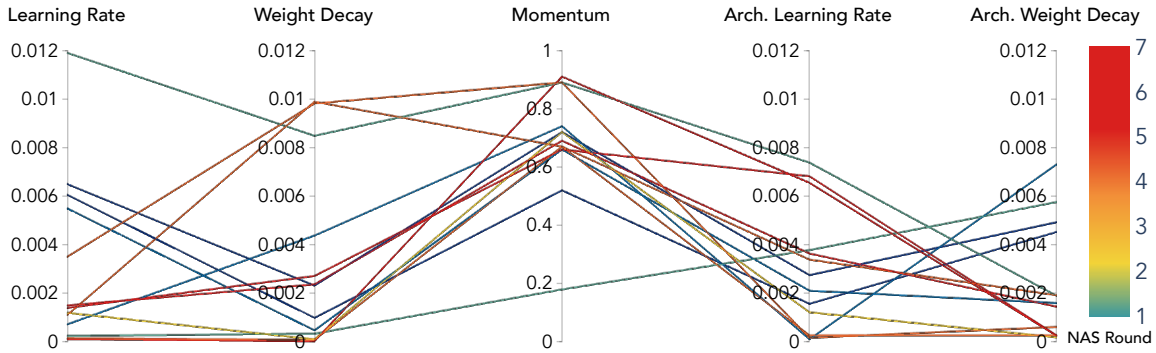

Figure 4: **FEATHERS adjusts hyperparameters over time**. The choices of hyperparameters are adapted during training to optimize the validation loss. In earlier stages (blue lines) higher learning rates are chosen whereas in later stages of training (red lines) lower learning rates are chosen. The figure shows hyperparameter-selections of three FEATHERS-runs on CIFAR-10.

FEATHERS preserves privacy. To demonstrate that FEATHERS provides privacy guarantees without sacrificing predictive performance, we performed classification on the fraud detection dataset. The same privacy budget $\epsilon \in \{1.25, 2.0, 3.5, 25, 250, 400, \infty\}$ was used for DP applied to the losses, model parameters and architecture parameters. Accounting was done via the RDP accountant. It is noteworthy that a privacy budget of ∞ corresponds to FEATHERS without DP. The search stage was performed for 100 communication rounds, all other parameters were set as above. Note that the dataset is heavily skewed (95% negative class, 5% positive class), we thus report F1-scores instead of accuracy. We further used oversampling of positive samples on the client-side to account for label-skew. Figure 2 visualizes the results for different privacy budgets

ϵ. For $\epsilon \geq 1$ we obtained a F1-score of approximately 0.77. This means, FEATHERS-DP performs equally well as FEATHERS as long as ϵ is chosen larger to be larger than 1. Decreasing ϵ adds more noise on the gradients which increases the privacy level while disturbing the gradient-signal. The effect of this is that for $\epsilon \leq 3.5$ we obtained a significant decrease of the F1-score.

In summary, FEATHERS-DP retains the performance of FEATHERS for appropriate ϵ. Finally, we emphasize that adding DP came with approximately 1.5-2 times longer runtimes on our setup. A reasonable trade-off to accommodate privacy considerations. The underlying reason is that for DP the gradient of each sample has to be manipulated, resulting in poorer parallel execution of automatic differentiation. Hence, there must be found a good trade-off between high performing models, training runtime and level of privacy in practice.