A DETAILS ON `FEDPN` ALGORITHM

A.1 TRAINING PROCEDURE

Recap, that in our framework, we have two stages – federated training and local training. After federated training, each of the clients has access to a global model. This model consists of the encoder, parametrized by ϕ, density model (normalizing flow in our case), parametrized by ψ, and classifier model with parameters θ. All these parameters were learned in a federated fashion, and effectively "felt" all the data, distributed over clients. In contrast, when we train local models (we train only flow and classifier from scratch, the encoder is fixed to the one after federated training), we use only local data.

To train the global normalizing flow in a federated fashion, we use the `FedAvg` standard approach, which begins with sending the current state of weights from the server to clients, and then after local training sending them back for aggregation. During inference on an unseen data object x at client i, we differentiate between θ, ψ and θ_i, ψ_i based on some uncertainty scores, computed using the density of $g_\phi(x)$ (density of embedding).

Since we draw inspiration from `NatPN`, we will refer to this particular instance of our framework as `FedPN` (Federated Posterior Networks). See Algorithm 1 for the detailed procedure.

Algorithm 1 FedPN

```
Require: Number of clients $b$, local datasets $D_i$, number of training federated rounds $R$, number of
    local iterations $L$, participation rate $\alpha$, normalizing flow prior $Pr$.
procedure TRAINFEDPN
    $\phi, \psi, \theta \leftarrow$ randInit()
    for $r = 0; r < R; r++$ do                                          ▷ Cycle over federated rounds
        $AC \leftarrow$ floor($\alpha b$)                                  ▷ Choose active clients
        for $i$ in $AC$ do
            $\psi_i \leftarrow \psi$
            $\theta_i \leftarrow \theta$
            for $l = 0; l < L; l++$ do                                  ▷ Cycle over local epochs
                for $(X_i, Y_i \in D_i)$ do                             ▷ Cycle over local dataset
                    $e(X_i) = g_\phi(X_i)$                              ▷ Compute embeddings
                    $\boldsymbol{\alpha}^{\text{post}}(X_i) = \boldsymbol{\alpha}^{\text{prior}} + \boldsymbol{f}_{\theta_i}\big(e(X_i)\big) p_{\psi_i}\big(e(X_i)\big)$
                    $L_1(X_i, Y_i) = L\big(y, \text{StopGrad}_{p_{\psi_i}(g(x))} \boldsymbol{\alpha}^{\text{post}}(X_i)\big)$
                    $L_2(X_i) = -\gamma \log p_{\psi_i}(X_i)$
                    $L_3(X_i) = -\lambda \mathcal{H}\big[\boldsymbol{Dir}(\boldsymbol{\mu} \mid \boldsymbol{\alpha}^{\text{post}}(X_i))\big]$
                    Minimize $L_1(X_i, Y_i) + L_2(X_i) + L_3(X_i)$
                end for
            end for
            Send active $\theta_i, \psi_i$ to server
        end for
        $\theta \leftarrow$ Mean($\theta_i$)
        $\psi \leftarrow$ Mean($\psi_i$)
    end for
end procedure
```

A.2 LOSS FUNCTION

Let us explore an asymptotic form of (5). For all $x > 0$, the following inequality holds:

$$\log x - \frac{1}{x} \leq \psi(x) \leq \log x - \frac{1}{2x}.$$

Recalling the update rule (1) and using the specific parameterization of $\alpha_c^{\text{prior}} = 1$ for all c, we conclude that all $\alpha_c^{\text{post}} > 1$. Hence, we can approximate $\psi(\alpha_c^{\text{post}}) \approx \log \alpha_c^{\text{post}}$.

To simplify the notation, we will use $z = g(x)$:

$$\begin{aligned}
L\big(y, \boldsymbol{\alpha}^{\text{post}}(x)\big) \approx \log\big(\alpha_0^{\text{post}}(x)\big) - \log\big(\alpha_y^{\text{post}}(x)\big) &= \log\left[\frac{\sum_{i=1}^{K} \alpha_i^{\text{prior}} + p(z)\sum_{i=1} f_i(z)}{\alpha_y^{\text{prior}} + p(z) f_y(z)}\right] \\
&= \log \sum_{i=1}^{K} \alpha_i^{\text{prior}} + \log\left[\frac{1 + p(z)\frac{1}{\sum_{i=1}^{K} \alpha_i^{\text{prior}}}}{\alpha_y^{\text{prior}} + p(z) f_y(z)} + 1 - 1\right] \\
&= \log \sum_{i=1}^{K} \alpha_i^{\text{prior}} + \log\left[1 + \frac{1 - \alpha_y^{\text{prior}} + p(z)\left(\frac{1}{\sum_{i=1}^{K} \alpha_i^{\text{prior}}} - f_y(z)\right)}{\alpha_y^{\text{prior}} + p(z) f_y(z)}\right] \\
&= \log \sum_{i=1}^{K} \alpha_i^{\text{prior}} + \log\left[1 + \frac{1 - \alpha_y^{\text{prior}}}{\alpha_y^{\text{prior}} + p(z) f_y(z)} + \frac{p(z)\left(\frac{1}{\sum_{i=1}^{K} \alpha_i^{\text{prior}}} - f_y(z)\right)}{\alpha_y^{\text{prior}} + p(z) f_y(z)}\right], \quad (9)
\end{aligned}$$

where we utilized the fact that $\sum_{i=1}^{K} f_i(z) = 1$ as a softmax vector, and exploited the posterior update rule (1).

Specifically, if we adopt the parameterization of (Charpentier et al., 2022; 2020), we should set all $\alpha_c^{\text{prior}} = 1$. This cancels the second term under the logarithm. Note that this term motivates $p(z)$ to concentrate mass on training examples, irrespective of the probability of the correct class $f_y(z)$.

A.3 PARAMETERIZATION OF NORMALIZING FLOW

In this subsection, we discuss the specifics of incorporating the normalizing flow into our pipeline. A critical advantage of the approach outlined in (Charpentier et al., 2022), in comparison with the one presented in (Charpentier et al., 2020), lies in the utilization of a single normalizing flow as opposed to learning individual flows for each class. This strategy proves to be highly effective particularly when the dimensionality of embeddings remains relatively low.

Nevertheless, we observed an increased complexity in the single flow's ability to approximate multiple modes corresponding to distinct classes in the context of higher-dimensional spaces. To address this problem, we propose the training of one flow per class, thereby aligning with the methodology of (Charpentier et al., 2020). However, we preserve the same parameterization as described in (Charpentier et al., 2022), effectively marginalizing the class labels. Subsequently, the density of the encoder network's embeddings is computed as follows:

$$p(z) = \sum_{c=1}^{K} p(z, c) = \sum_{c=1}^{K} p(z \mid c)\, p(c),$$

where z represents the embedding (output of the encoder network), $p(z \mid c)$ is the density of z with respect to the normalizing flow with index c, and $p(c)$ denotes the prior probability of class c.

We estimate the prior class probabilities based on each client's data, attaching the relative proportion of the class. If a particular class is not represented in a client's data, the corresponding class probability is set to zero, thus the flow's parameters are not updated. It is worth noting that as the global model has no access to the client's data, we set a uniform distribution over the classes.

Note, that in principle, we can use any density model in the approach (e.g. considering a Gaussian Mixture Model with a Gaussian, fitted to each class). However, this study is beyond the scope of the paper.

A.4 HYPERPARAMETERS SELECTION

In this section, we discuss the selection process for the hyperparameters associated with our framework. To ease the reading experience, we will consolidate all the hyperparameters into a tabulated format. For the summary, see Table 4.

Dataset	Batch size	Optimizer	Learning rate	Momentum	Architecture	Logprob weight γ	Entropy weight λ	Federated rounds	Local epochs
MNIST	64	SGD	0.01	0.9	LeNet-5	0.01	0.0	100	10 batches
FashionMNIST	64	SGD	0.01	0.9	LeNet-5	0.01	0.0	100	10 batches
MedMNIST-A	64	SGD	0.01	0.9	LeNet-5	0.01	0.0	100	10 batches
MedMNIST-C	64	SGD	0.01	0.9	LeNet-5	0.01	0.0	100	10 batches
MedMNIST-S	64	SGD	0.01	0.9	LeNet-5	0.01	0.0	100	10 batches
CIFAR10	64	SGD	0.01	0.9	ResNet-18	0.01	0.0	100	10 batches
SVHN	64	SGD	0.01	0.9	ResNet-18	0.01	0.0	100	10 batches

Table 4: Summary of the hyperparameters, used for the training of our framework.

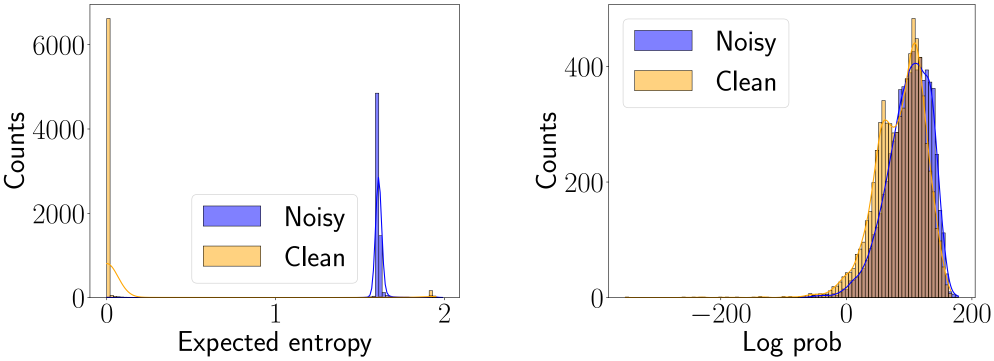

Figure 3: **Left**: The histogram depicts aleatoric uncertainty, represented as the expected entropy of the predictive distribution. It was computed using the formula specified in equation (4). **Right**: The histogram demonstrates epistemic uncertainty. It is computed using the logarithm of the density of the embeddings.

When the federated training process is concluded, we keep the resulting encoder for all the client models as fixed and reinitialize the flow and classifier models. Subsequently, we train only the flow and the classifier models, using only the data specific to a given client. This training phase lasts for 10 local epochs using Adam optimizer with a learning rate of $1e-3$ and the same γ as in the federated stage.

B ADDITIONAL EXPERIMENTS

B.1 DISTINCTION BETWEEN ALEATORIC AND EPISTEMIC UNCERTAINTIES

In the main discussions of this paper, we directly considered only two parts of our framework, resulting in a "switching" model, where one could abstain from the prediction of the local model and use the global model instead, or abstain from the prediction at all (if the certainty of global model is low).

In this part of the paper, we study another part of the pipeline – the ability to abstain from the prediction if aleatoric uncertainty (label noise or input ambiguity) is sufficiently big. Typical benchmark datasets like the ones we used are carefully curated and are found to have almost no label noise, as underscored by (Kapoor et al., 2022). Consequently, illustrating aleatoric uncertainty based solely on the information derived from these datasets can pose significant challenges.

Nevertheless, to facilitate the comprehensive execution of our proposed pipeline, we manually introduce artificial label noise. This strategic introduction serves a twofold purpose – not only does it effectively demonstrate our model's capability to discern between aleatoric and epistemic uncertainties, but it also showcases that our model is unlikely to transition to a global model in the presence of high aleatoric uncertainty.

To conduct this examination, we change the labels of five classes – specifically, classes 5, 6, 7, 8, and 9 – within the MNIST dataset. The remainder of the classes, namely classes 0, 1, 2, 3, and 4, are retained in their original, unaltered state, thereby maintaining a degree of purity and control within our experiment.

Dataset	FedAvg	FedRep	PerFedAvg	FedBabu	FedPer	FedBN	APFL	**FedPN**
MNIST	87.6	99.3	99.3	**99.6**	99.5	74.2	77.9	98.4
FashionMNIST	66.4	95.3	95.1	**95.8**	**95.8**	56.3	77.0	84.3
MedMNIST-A	58.1	96.9	96.2	97.3	**97.7**	47.8	98.0	96.2
MedMNIST-C	49.5	93.0	91.7	95.0	95.2	50.0	**95.4**	94.4
MedMNIST-S	38.9	87.4	86.7	90.5	90.9	40.5	**91.8**	86.9
CIFAR10	27.6	81.2	73.3	**84.1**	**84.1**	35.2	62.4	59.1
SVHN	80.6	94.7	93.4	94.9	**95.4**	74.4	80.5	87.1

Table 5: InD Data: Accuracy scores for in-distribution data. Values in **bold** are the best results.

Dataset	FedAvg	FedRep	PerFedAvg	FedBabu	FedPer	FedBN	APFL	**FedPN**
MNIST	82.3	0.0	50.0	61.8	29.4	65.0	58.9	**98.3**
FashionMNIST	55.2	0.0	22.9	22.4	15.2	51.8	34.0	**78.2**
MedMNIST-A	47.5	0.0	12.5	8.1	12.6	43.3	49.0	**94.9**
MedMNIST-C	45.5	0.0	2.4	13.6	10.9	43.3	53.3	**88.7**
MedMNIST-S	34.8	0.0	3.2	5.7	6.0	38.9	34.0	**75.5**
CIFAR10	23.3	0.0	0.0	1.4	0.6	28.6	15.0	**28.8**
SVHN	**76.7**	0.0	9.5	11.2	6.5	71.9	42.5	62.2

Table 6: OOD Data: Accuracy scores for out-of-distribution data. Values in **bold** are the best results.

The results of this experiment are presented in Figure 3. We see, that the aleatoric uncertainty, evaluated as an expected entropy of predictive distribution, drastically differs for noisy labels and for clean ones. Apparently, the threshold for the separation can be easily chosen using this histogram (e.g. the value of 1). Contrary to the measure of aleatoric uncertainty, the scores of epistemic uncertainty (here it is the logarithm of the density of features) can not distinguish between noisy and clean labels.

To conclude, the proposed framework effectively allows us to distinguish between noisy labels through the disentanglement of uncertainties into aleatoric and epistemic parts. This allows us to abstain from the prediction (when the label noise), or delegate inference to the global model (when input is out-of-distribution).

B.2 COMPUTATIONAL OVERHEAD

One may worry that the introduction of the extra density model will make the approach prohibitively expensive. In fact, everything depends on the capacity of the chosen density model. Note, that in our case we are using one of the most simplest normalizing flows, namely Radial flow. For our experiments, we considered LeNet5 backbone (with 55k parameters), while the stack of normalizing flows (30 flows) had 5k parameters overall, so the computational overhead is very mild.

B.3 SEPARATE TABLES FOR IND AND OOD.

In this section, we represent Table 2 as two separate tables: Table 5 for benchmarking on InD data and Table 6 for benchmarking on OOD data. We clearly observe that FedPN shows superior results on OOD data while being competitive on in-distribution samples.

B.4 ABLATION STUDY FOR THRESHOLD SELECTION

This experiment focuses on an ablation study of the selection of a threshold for switching between Local and Global models, based on the epistemic uncertainty of the Local model. We assessed the performance by measuring the accuracies of Local, Global, and Switch models on both InD and OOD data. We then calculated epistemic uncertainty scores, specifically the entropy of the predictive Dirichlet distribution (see Equation 3), on a calibration subset of the data. It's important to note that

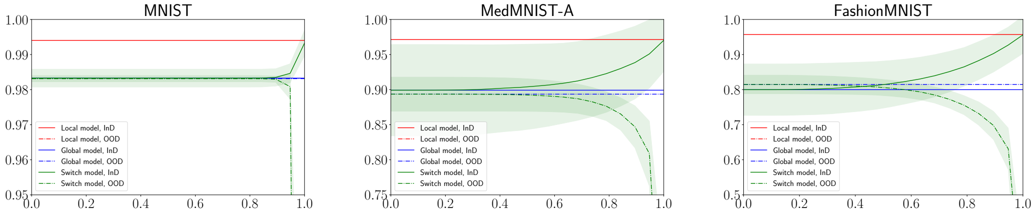

Figure 4: Performance comparison of Switch, Local, and Global models with varying switching thresholds. Note that the Local model's accuracy is almost zero and does not appear within the considered range of y-axis values.

this subset comes from the same distribution as the training data. The threshold selection, therefore, is a nuanced process in this context. Given the absence of explicit OOD data for more precise threshold tuning, we pretend that a portion of the calibration data is an OOD and select a corresponding quantile of epistemic uncertainty scores as our threshold.

The result of this experiment are shown in Figure 4. Here, the x-axis represents the quantile chosen for the threshold, while the y-axis displays the resulting accuracy.

We see that for some datasets, such as MNIST, the threshold selection is straightforward. The Switch model's performance on InD and OOD aligns rapidly, and its accuracy is relatively insensitive to threshold variations.

In contrast, datasets like MedMNIST-A and FashionMNIST demand more careful threshold selection. Nevertheless, even for these datasets, a reasonable choice (e.g., quantile ($q \in [0.8, 0.9]$, which correspond to $10 - 20\%$ of OOD in calibration data) yields satisfactory results.

Therefore, while threshold selection involves subjectivity, experiment shows that there is a "reasonable corridor" for threshold values that do not significantly compromise the accuracy of the resulting Switch model.

B.5 THE EFFECT OF OUR STOPGRAD MODIFICATION

In this section, we conduct an empirical study to understand how our modified loss function impacts training under conditions of aleatoric uncertainty in the data. To simulate this, we introduce label noise into the MNIST dataset. Specifically, labels for half of the classes (5-9) are randomly swapped amongst themselves. We then set up a Federated Learning scenario with 20 clients, where each client has data from no more than three classes.

We initiate the federated training process using either our modified loss function (see Equation 8) or the standard Bayesian loss function (see Equation 6), keeping the number of training epochs consistent across both methods.

Figure 5 presents normalized histograms of the resulting $\log p(z)$ values for global models. This visualization reveals that when faced with aleatoric (label) noise, the model trained with the standard loss function tends not to assign significant probability mass to the noisy examples, even though these examples are part of its training distribution. In contrast, the model trained with our proposed loss function is incentivized to allocate probability mass to these examples. This outcome aligns with the expectation that $p(z)$ represents the density of training samples.

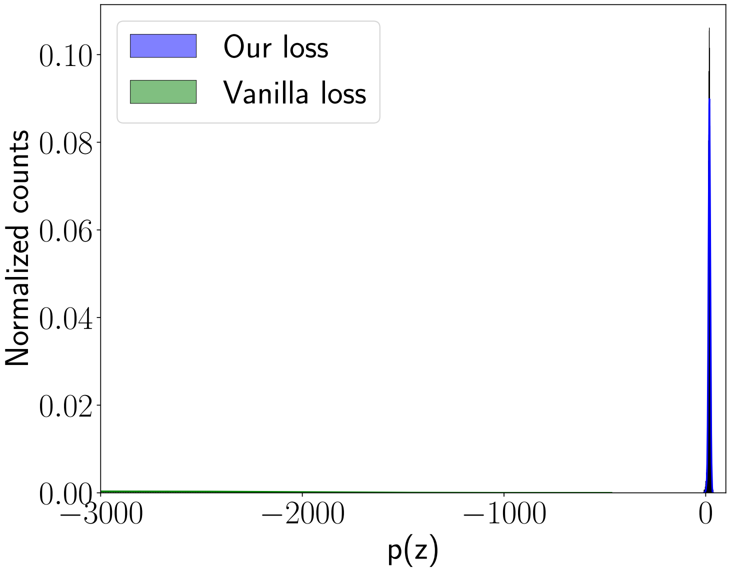

Figure 5: Normalized histogram of $\log p(z)$ for models trained with the standard Bayesian loss 6 and our modified loss 8.