Supplementary Materials for

Temporal speckle enables probabilistic weights for uncertainty-aware photonic AI

F. Brückerhoff-Plückelmann[1*], H. Borras[2*], S. U. Hulyal[3*], L. Meyer[1], X. Ji[3], J. Hu[3], J. Sun[3], B. Klein[2], F. Ebert[1], J. Dijkstra[1], L. McRae[1], P. Schmidt[1], T. J. Kippenberg[3†], H. Fröning[2], W. Pernice [1†]

[*]These authors contributed equally to this work.

[†]Corresponding author. Email: wolfram.pernice@kip.uni-heidelberg.de, tobias.kippenberg@epfl.ch

Contents

Supplementary Text

Probability density of the measured optical fluctuations

Amplified spontaneous emission produces a chaotic optical field whose frequency components have random relative phases. Square-law photodetection converts their mutual beating into temporal power fluctuations. We consider a temporally varying input sequence encoded by the electro-optic modulator. The input sequence has zero temporal mean, such that, within the linear modulation regime, the time-averaged detector signal remains constant over the acquisition window.

For a fixed input value, repeated realizations of the detector signal before high-pass filtering follow a gamma distribution,

$$f(z;\mu,M) = \frac{M^M}{\mu\,\Gamma(M)}\left(\frac{z}{\mu}\right)^{M-1}\exp\!\left(-\frac{Mz}{\mu}\right), \qquad z \geq 0, \tag{S1}$$

where μ is the conditional mean signal before high-pass filtering and M is the effective number of independent temporal coherence cells contributing to one electrical measurement [16]. The parameter M is not restricted to integer values. This parameterization gives $\mathrm{Var}(z) = \mu^2/M$, such that the relative standard deviation decreases as $M^{-1/2}$.

The photoreceiver is AC-coupled and removes the slowly varying component Δx of the detector signal. We denote the measured signal after high-pass filtering by x and its conditional mean by $\bar{x}$. The corresponding pre-filter mean is therefore $\mu = \bar{x} + \Delta x$, and the probability density of the optical fluctuations after high-pass filtering is

$$f_{\mathrm{PD}}(x;\bar{x},M,\Delta x) = f(x+\Delta x;\bar{x}+\Delta x,M)\,, \qquad x \geq -\Delta x. \tag{S2}$$

For the zero-mean input encoding used here, Δx corresponds to the constant detector contribution at the electro-optic-modulator bias point. The input-dependent component changes $\bar{x}$ but does not change the baseline removed by the high-pass filter.

The electronic readout additionally contributes independent, zero-mean Gaussian noise,

$$g(v;\sigma_{\mathrm{el}}) = \frac{1}{\sigma_{\mathrm{el}}\sqrt{2\pi}}\exp\!\left(-\frac{v^2}{2\sigma_{\mathrm{el}}^2}\right), \tag{S3}$$

where σ_{el} is the electronic-noise standard deviation. The measured photodetector voltage distribution is the convolution of the shifted gamma distribution and the electronic-noise distribution,

$$p(x;\bar{x},M,\Delta x,\sigma_{\mathrm{el}}) = \int_0^\infty f(z;\bar{x}+\Delta x,M)\,g(x-z+\Delta x;\sigma_{\mathrm{el}})\;\mathrm{d}z. \tag{S4}$$

For the measurements in Fig. 2, we apply no temporal input modulation. The optical statistics are therefore stationary and, once the high-pass transient has decayed, the measured distribution is

centred around zero, such that $\bar{x} = 0$. The removed component Δx then equals the mean of the stationary pre-filter signal. We determine σ_{el} from measurements with the optical input blocked and keep it fixed for all fits. Because we hold the integrated optical power constant while varying the channel bandwidth, we use a common Δx for all bandwidths measured at the same optical power. We further constrain Δx to scale linearly with the measured optical power across the three power settings. We fit M independently at each operating point. The extracted values increase approximately linearly with optical bandwidth and remain largely independent of optical power, as shown in Fig. 2d.

Physical Computation Model

The photonic accelerator implements a probabilistic one-dimensional convolution, but its analogue operation depends on several physical parameters. These include the bias-point transmission b of the electro-optic modulator (EOM), which couples the analogue electronic input to the optical signal; the encoding range r around the bias point, which depends on the π-voltage of the EOM and the applied electronic voltage swing; and the electronic-noise standard deviation σ_{el}. During computation, the oscilloscope clips the detector voltage to $\pm 110\,\mathrm{mV}$ and maps this interval to the normalized output range $[-16, 16]$. We choose this range because the maximum mean output lies within $[-9, 9]$ for a convolution with inputs in $[-1, 1]$ and a nine-component kernel with weights in $[0, 1]$. We set the EOM bias-point transmission to $b = 0.4$ via the EOM calibration measurement.

We determine the encoding range r and the electronic ground noise σ_{el} by fitting the following computation model to measured output distributions with different optical powers and bandwidths for each frequency bin. For an input vector x with $x_i \in [-1, 1]$ the mean optical power after the EOM at time step i is:

$$\langle P_{\mathrm{EOM}} \rangle_i = \sum_{k=1}^{9} \langle P_{k,\mathrm{weight}} \rangle \left(b + r \cdot x_i \right) \tag{S5}$$

Here $\langle P_{\mathrm{k,weight}} \rangle$ is the mean optical power of each frequency bin. Since the insertion loss of all components is constant over time, it's already implicitly considered here.

After the chirped waveguide Bragg grating applies the frequency-dependent group delay, the mean optical power at the photodetector at time step $i + \tau$ is

$$\langle P_{\mathrm{PD}} \rangle_{i+\tau} = \mathrm{conv1d}\big(b + rx,\, \left\langle P_{\mathrm{weight}} \right\rangle\big)_i \,, \tag{S6}$$

where τ is the total sample delay between input sample i in the arbitrary-waveform-generator waveform and the corresponding output sample in the oscilloscope waveform. Because the highest-frequency channel experiences the smallest group delay, we reverse the weight order when mapping the kernel coefficients to the frequency channels. We use the PyTorch convention for conv1d, which differs from the NumPy convolution convention by a reversal of the weight vector.

The analogue readout electronics applies a high-pass filter with a cut-off frequency of 45 kHz while amplifying the detector output. For the zero-mean input encoding used here, the high-pass filter removes the constant bias contribution from the mean measured voltage. We use a constant scaling factor α to account for the $1000\,\mathrm{V\,W^{-1}}$ conversion gain of the photodetector and transimpedance amplifier, together with the rescaling applied after analogue-to-digital conversion. The mean output distribution is therefore

$$\langle y \rangle = \mathrm{conv1d}\big(x,\ \alpha r \left\langle P_{\mathrm{weight}} \right\rangle\big)\,. \tag{S7}$$

The accelerator can thus implement convolution operations with signed inputs and positive weights by programming the optical power in each frequency channel.

Because the power fluctuations in different frequency channels are uncorrelated, their variances add at the photodetector. The variance of the output distribution is

$$\mathrm{Var}(y) = \mathrm{conv1d}\left(\left(x + \frac{b}{r}\right)^2,\ \left(\alpha r \sigma_{\mathrm{weight}}\right)^2\right) + \sigma_{\mathrm{el}}^2, \tag{S8}$$

where the squares are evaluated element-wise. Although the high-pass filter removes the bias-point contribution from the mean output, the bias still scales the high-frequency optical fluctuations and therefore contributes to the output variance. The standard deviation σ_{weight} of each probabilistic weight depends on the optical power and bandwidth of the corresponding frequency channel. We use $r = 0.26$ and $\sigma_{\mathrm{el}} = 0.52$ to emulate the physical accelerator during hardware-aware training.

Error model

Hardware-aware training requires a robust and computationally efficient measure of analogue computation accuracy. We model the residual deviations of the physical accelerator by adding Gaussian-distributed perturbations to the output mean and standard deviation predicted by the floating-point computation model:

$$\begin{aligned} \bar{y}_{\mathrm{meas}} &= \bar{y}_{\mathrm{fp}} + \lambda_{\mathrm{mean}}\,\mathcal{N}(0,1), \\ \mathrm{std}(y_{\mathrm{meas}}) &= \mathrm{std}\big(y_{\mathrm{fp}}\big) + \lambda_{\mathrm{std}}\,\mathcal{N}(0,1). \end{aligned} \tag{S9}$$

The Gaussian perturbations are sampled independently for the output mean and standard deviation. To reproduce the experimentally observed hardware deviations during training, we use $\lambda_{\mathrm{mean}} = 0.11$ and $\lambda_{\mathrm{std}} = 0.053$.

Delay and skew calibration

Accurate calibration of the temporal offset between the symbols transmitted by the digital-to-analogue converter (DAC) and those recorded by the analogue-to-digital converter (ADC) is essential for correct convolution alignment. Because the propagation delay depends on optical frequency, we

perform the calibration at the system centre frequency of 194 THz. We disable all other frequency channels, such that the system implements an identity operation rather than a temporal convolution.

We first transmit a random input vector containing 80 000 components and determine the optimal integer sample delay at the ADC, corresponding to the number of samples between the oscilloscope trigger and the recorded output waveform. Figure S1a shows the deviation between the transmitted and received symbols as a function of sample delay and exhibits a clear minimum. The minimum extends across more than one sample because each symbol is represented by three samples, which relaxes the timing requirements.

We then calibrate the sub-sample delay by sweeping the oscilloscope skew, which controls the temporal offset between the trigger and ADC acquisition. We vary the skew in steps of 1 ps and determine the optimal sample delay at each setting (Fig. S1b). The resulting plateau-like behaviour reflects the 12.5 ps sample duration at the 80 GSa s^{-1} acquisition rate. We set the skew to the centre of the plateau to maximize tolerance to residual timing errors.

Iterative weight programming

Rather than programming each probabilistic weight independently, we optimize all frequency channels simultaneously using an iterative feedback procedure. In each iteration, the photonic accelerator evaluates probabilistic convolutions for a random input vector of length 300, and we acquire 2500 realizations of each output distribution. We fit the physical computation model to the measured distributions to estimate the currently programmed weight means μ_p and standard deviations σ_p.

We then update the channel power P and optical bandwidth $\Delta\nu$ according to their deviations from the target mean μ_t and standard deviation σ_t:

$$\begin{aligned} \Delta P &= \frac{\mu_\mathrm{t} - \mu_\mathrm{p}}{\alpha}, \\ \Delta(\Delta\nu) &= -\frac{\sigma_\mathrm{t} - \sigma_\mathrm{p}}{\beta\mu_\mathrm{p}}. \end{aligned} \tag{S10}$$

We apply these updates independently to each frequency channel. The calibration factors α and β approximate the responses of the programmed mean to optical power and of the standard deviation to optical bandwidth, respectively.

We scale the updates using a cosine-annealed factor that decreases from 1.5 in the first iteration to 0.5 in the final iteration. We use ten iterations to program the complete probabilistic kernel. Throughout the procedure, we constrain the channel bandwidth to the range 25 GHz to 150 GHz to limit pulse-shape distortion arising from intrachannel dispersion.

Stochastic variational inference

Stochastic variational inference (SVI) provides a natural framework for learning the parameterized probability distributions represented by the photonic accelerator. SVI approximates an analytically intractable posterior distribution with a tractable variational distribution whose parameters are optimized by gradient descent [48, 49]. This approach is compatible with the photonic hardware because the encoded weight distributions are parameterizable and their likelihoods can be evaluated efficiently.

We use the Bayes-by-Backprop formulation of SVI [6], which applies the reparameterization trick to express stochastic sampling as a deterministic transformation of the learned distribution parameters and an auxiliary noise variable. This formulation enables gradients to propagate through stochastic network parameters during backpropagation.

We construct the Bayesian neural network in the same manner as a deterministic deep neural network, using stacked fully connected and convolutional layers, also see Fig. S2. We replace selected deterministic parameters with probability distributions while retaining the remaining parameters as point estimates. Gradient-based optimization then jointly learns the deterministic parameters and the parameters of the probabilistic weights.

Bayesian Neural Network Design

At the core of the Bayesian Neural Network implementation is the Probabilistic Programming Language: Pyro. The language is designed with SVI as its central component and provides a high-performance interface to GPU accelerated Bayesian inference, by building on top of PyTorch. Within Pyro we implement the custom probabilistic software component, modeling the photonic Bayesian machine, as a Pyro Module. Here we put the primary design focus on flexibility to adapt to the hardware as it changes throughout the co-design process. To support SVI for the photonic Bayesian machine the operator implements both Eq. S7 and Eq. S8 on a PyTorch level. While learning the deterministic weights required for Eq. S7 is trivial using the automatic gradient feature of PyTorch, we apply the following equation to arrive at σ_{weight} to implement Eq. S8:

$$\sigma_{\text{weight}} = \left(\sigma_{\text{rel}} \cdot (\sigma_{\max} - \sigma_{\min}) + \sigma_{\min}\right) \cdot \langle P_{\text{weight}} \rangle \tag{S11}$$

Here σ_{rel} is the learnable probabilistic parameter, while σ_{min} and σ_{max} represent the bounds within which the hardware can represent the parameter. The advantage of this approach is that we do not need to account for changes to the hardware in the SVI learning process, as this is handled by flexibly setting the constant bound parameters. As σ_{rel} is a learnable probabilistic parameter, we implement it as a Pyro parameter. A uniform prior bound between [0,1] effectively ensures that the posterior is bounded to the same range. We then utilize a posterior of randomly initialized Delta distributions, which change their parameters to approximate the true posterior during training. During training the

expected Gaussian distribution of the hardware is sampled, which is computed from the results of Eq. S7 and Eq. S8. To model differences between the floating-point representation of the software and the actual hardware we primarily use straight-through gradient estimators. Their efficacy is well explored in the context of Quantization Aware Training [50]. From a basic functionality perspective, the idea is to apply disturbances (such as quantization) to the floating-point variables during the forward path of a neural network, while disregarding them during the backward path. This allows for the computation of the full gradient during the backwards path, even if the observed disturbances are generally non-differentiable. Using this principle we model the following hardware properties:

- Discretization of the DAC: quantization of fixed scale to eight bits (using QAT based on Brevitas)
- Clipping of the ADC: clamping of the operator output to: [-16, 16] (using a custom straight-through gradient estimator)
- Programming precision of the hardware for the parameters of σ_{weight} and $\langle P_{\text{weight}} \rangle$, as described in Eq. S9, using a straight-through gradient estimator

To interface with the photonic hardware, we develop a PyTorch compatible interface. As the hardware executes a 1D convolution, the API is made to be compatible with the functional implementation of a 1D convolution in PyTorch. This simplifies development, as no access to the hardware is required during initial development, only for the later evaluation. To this end the custom software operator implements a second execution path to switch from software to hardware execution, without the need for changes to the BNN model. The hardware execution path in particular encompasses the conversion of the 2D convolution to a 1D convolution. To enable data reuse and fast execution times, we implement a custom conversion function with numba, which converts the inputs σ_{weight} and $\langle P_{\text{weight}} \rangle$ from 2D to 1D and back. This is required, as such conversions are not possible with high data reuse within PyTorch. Here, we ensure high data reuse and efficient execution by making use of the just-in-time compilation capabilities of the software package numba. Similarly, after the hardware execution, the 1D outputs are converted back to a 2D tensor, for further processing within the rest of the model. The architecture design of the BNN requires careful consideration of the photonic Bayesian machine. As the hardware highly values grouped convolutions over standard ones, we deploy an architecture search to find BNN models most suitable for the photonic hardware. We in particular consider DenseNet-like skip-connections [18] to increase the robustness of the resulting architecture with respect to noise [51]. Along with this, we employ a custom type of Depthwise Separable convolutions as described by MobileNet [19]. This again maximizes the number of grouped convolutions. However, in contrast to standard DWS convolutions, we perform the pointwise convolution as a 1D convolution along the channel dimension. A survey of different architecture candidates is shown in Fig. S3, while the final resulting architecture is described in the main manuscript. As a result of hyperparameter optimization we also find an optimal activation

function for the network, out of a selection of many. Notably this activation function is a modification to LeakyReLU, as proposed by Tempczyk et. al [52], where the slope of the negative domain is set to -0.5, which allows for more Gaussian-like posteriors and lower expected calibration errors.

Bayesian Neural Network Training

After designing the BNN architecture as above, we train it using the standard SVI training loop of Pyro. Along with dataset preprocessing in the form of normalization for all datasets we also employ data augmentation for the MedMNIST dataset [24]. The official reference implementation of TrivialAugment [53] is used for data augmentation. This efficiently allows us to control the amount of data augmentation in a range of 0 to 32. Notably for the Dirty-MNIST [54] testcase we only train on MNIST, which is a conceptually more difficult task than the standard task [15, 25, 26], in which one trains on the training splits of both MNIST and Ambigious-MNIST at the same time. For our network the capability to distinguish aleatoric and epistemic uncertainty thus purely arises from the Bayesian Neural Network. For the MedMNIST task the Bayesian Neural Network is then trained with the following hyperparameters, which are the result of an extensive Hyperparameter search using ASHA:

- Training Dataset: bloodmnist
- Filtered_class: 2
- Data augmentation factor: 17
- train_batch_size: 256
- num_epochs: 2500
- num_train_samples: 3
- Loss function: TraceMeanField_ELBO
- learning rate: 0.000528
- weight_decay: 0.
- Optimizer: AdamW
- KL annealing schedule: linear
- KL max: 0.327
- Activation function: LeakyReLU -0.5

The training process is shown exemplary for the MedMNIST dataset in Fig. S4. Here we observe overall good convergence behavior. For Dirty-MNIST specifically we additionally employ a convergence aiding training procedure, in which we continuously scale up stochasticity in the network over the first 200 epochs. As this procedure is a variation of noisy training (NT) widely used for deterministic neural networks on analog hardware, we denote it here as incremental NT. For the Dirty-MNIST task a similar Hyperparameter search was conducted, with the following results, notably we utilize the schedulefree [55] version of RAdam[56]:

- Training Dataset: MNIST
- Data augmentation factor: 0
- train_batch_size: 256
- num_epochs: 2500
- num_train_samples: 4
- Loss function: TraceMeanField_ELBO
- learning rate: 0.00270
- weight_decay: 0.
- Optimizer: RAdamScheduleFree
- KL annealing schedule: linear
- KL max: 0.000211
- Activation function: LeakyReLU -0.5
- NT num_epochs: 200
- NT schedule: linear then plateau

Similarly to Fig. S4, the training process for Dirty-MNIST is shown in Fig. S5. We note that while the training shows similarly good convergence behavior to the MedMNIST results, we find differences in other aspects of the training: The learning curves are overall smoother, since the chosen optimizer reacts better to the stochastic environment of SVI in general. We further observe, particularly for the AUROC, that the training process is highly dynamic, which is mostly, but not completely addressed by KL annealing.

Comparison between Hardware and Software Results

To compare how well the software model matches the photonic Bayesian machine we investigate how the key metrics for MedMNIST change over multiple samples for both the hardware and software setting. These findings are shown in Fig. S6. We observe that given enough samples the software model matches the photonic hardware extremely well. While both models show similar trends with the number of samples drawn, it appears that a minimum of ten samples is required to adequately capture the stochasticity of both SVI software model and the SVI hardware.

Subsampling Dirty-MNIST

For Dirty-MNIST we only use 1/5th of full test dataset during prediction and equally sample randomly from all three test datasets (MNIST, Ambiguous-MNIST and Fashion-MNIST). To verify that even with the reduced dataset size we achieve comparable performance to the full datasets, we perform an ablation study on the software model. We assume this to be a good proxy for overall performance, due to the previously shown very good matching between software and hardware at 10 samples. The measured results are shown in Tab. S1. From this we conclude that subsampling has no significant impact on the final hardware results.

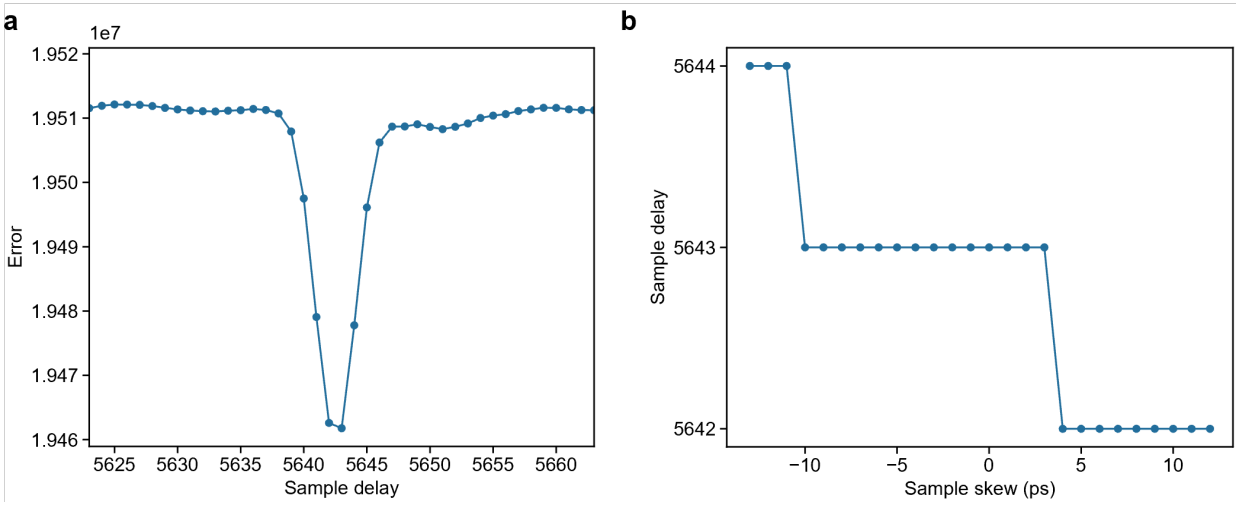

Fig. S1: Delay and skew calibration. **a**, We calibrate the sample delay between the triggering of the oscilloscope and the received data. There are two similarly good sample points as we use three samples to encode one symbol. **b**, We sweep the skew between the triggering of the oscilloscope and the capturing of the ADC to compensate for sub-sample delay. The optimal sample delay has a plateau-like shape given by the length of one sample of 12.5 ps. We choose the skew such that we are in the middle of a plateau.

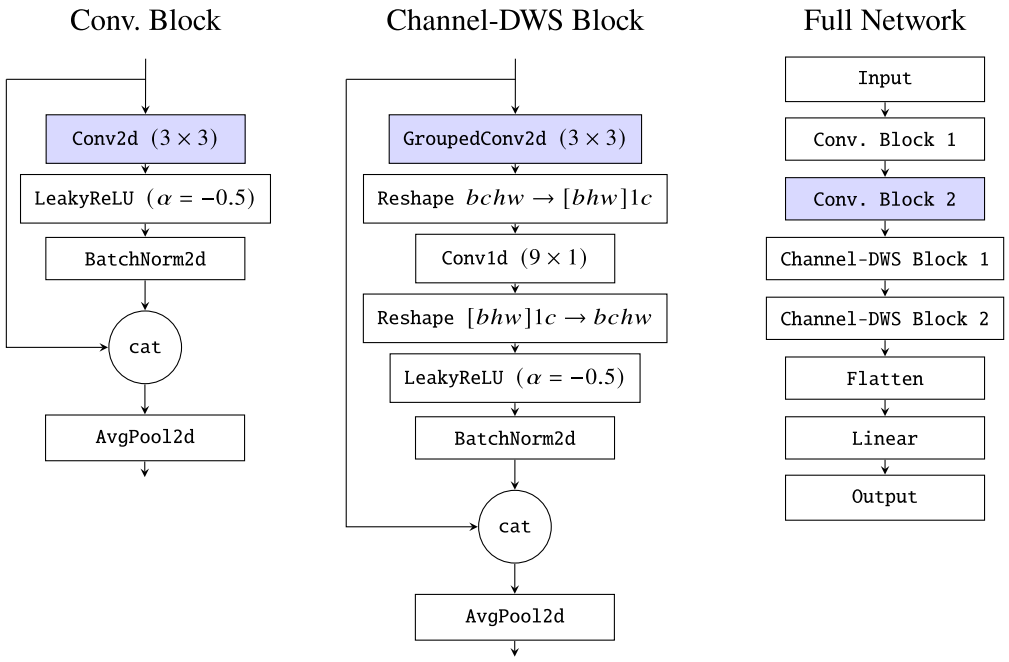

Fig. S2: Hybrid photonic-digital Bayesian neural network. We design a hybrid convolutional neural network for image classification, computing the deterministic layers digitally and offloading the probabilistic convolutions, marked in blue, to the photonic accelerator via its PyTorch interface. Two different types of convolutional blocks with skip connections are used to stack in total six convolutional layers, followed by a final linear layer. The size of the final linear layer depends on the number of different classes within the training dataset. All skip connections use concatenation over the channel dimension, as described by [18]. The DWS convolution operates as two convolutions, where the first is fully grouped, while the second one is a 1D Convolution over the channel dimension. Thus, we reorder the tensor shape of *Batch (b)*, *Channel (c)*, *Height (h)*, *Width (w)* to move the channels into the last dimension, while simultaneously moving all other parts into the first dimension and inserting a place holder dimension, denoted by *1*.

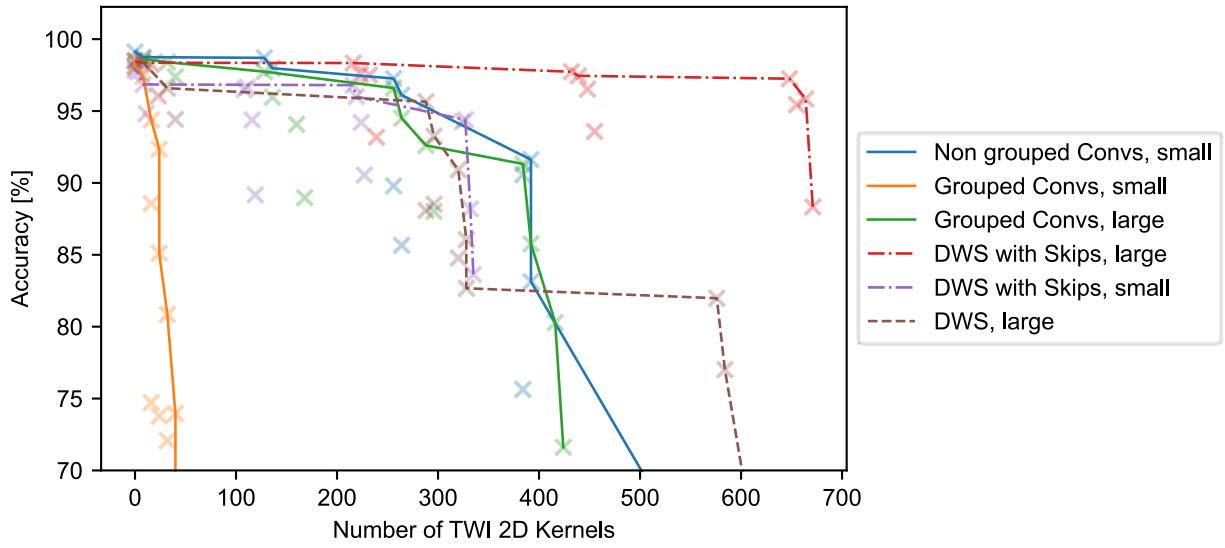

Fig. S3: Architecture Search. For multiple architecture candidates we plot their final test accuracy on MNIST against the number of photonic kernels in the network, when enabling different combinations of layers as probabilistic. Ideally, we try to maximize for both at the same time. For each candidate architecture we then draw a pareto frontier, making comparisons possible. Multiple conclusions can be drawn here: Grouped convolutions require more kernels compared to full convolutions to achieve the same accuracy; Depthwise Seperable (DWS) convolutions on their own make little difference to just grouped convolutions, however combining both DWS convolutions and skip connections enables smaller networks at the same accuracy, while scaling even better to larger networks. From these results we then choose the architecture with dominant pareto frontier for the final BNN architecture.

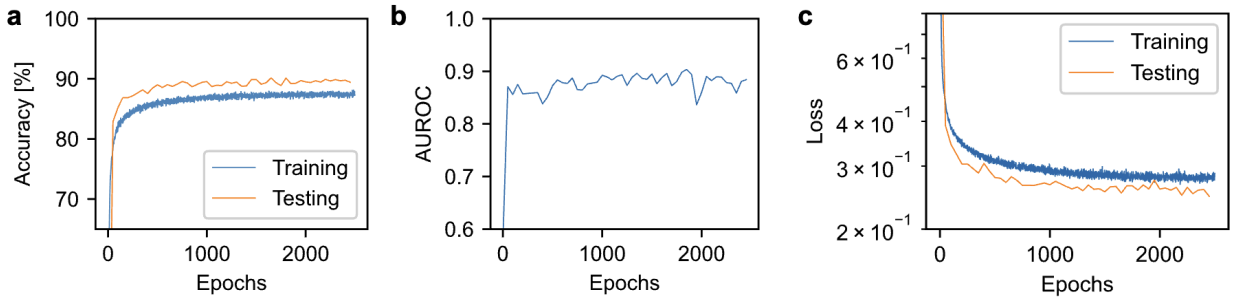

Fig. S4: Bayesian Neural Network Training Process for MedMNIST. a, The accuracy shows good convergence to an expected maximum. **b**, Similarly to the accuracy, the AUROC values against OOD data converge towards an optimal point over training. **c**, The loss curve during training shows very good convergence. We observe strong monotonicity and very little overfitting.

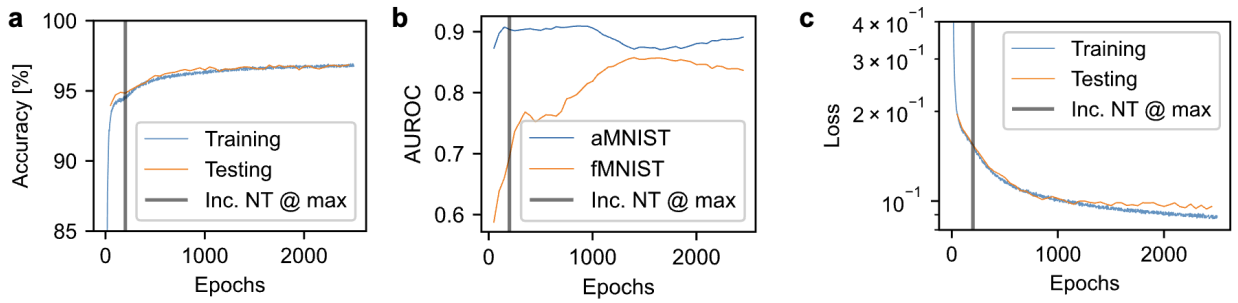

Fig. S5: Bayesian Neural Network Training Process for Dirty-MNIST. a–c In general, the training behaves like MedMNIST training in Fig. S4, however, we observe a significant impact of incremental noisy training (NT), where the end of the linearly increasing schedule is denoted as a black bar. Here we see that both the Accuracy and AUROC curves show significant changes and appear to benefit from an initial convergence to a high level. Interestingly the choice of optimizer results in an overall very smooth learning behavior, even under changing noise conditions and especially for the loss curve. Notably, we observe that the training process for Dirty-MNIST is much more dynamic for OOD detection than for MedMNIST. However, we still observe good convergence.

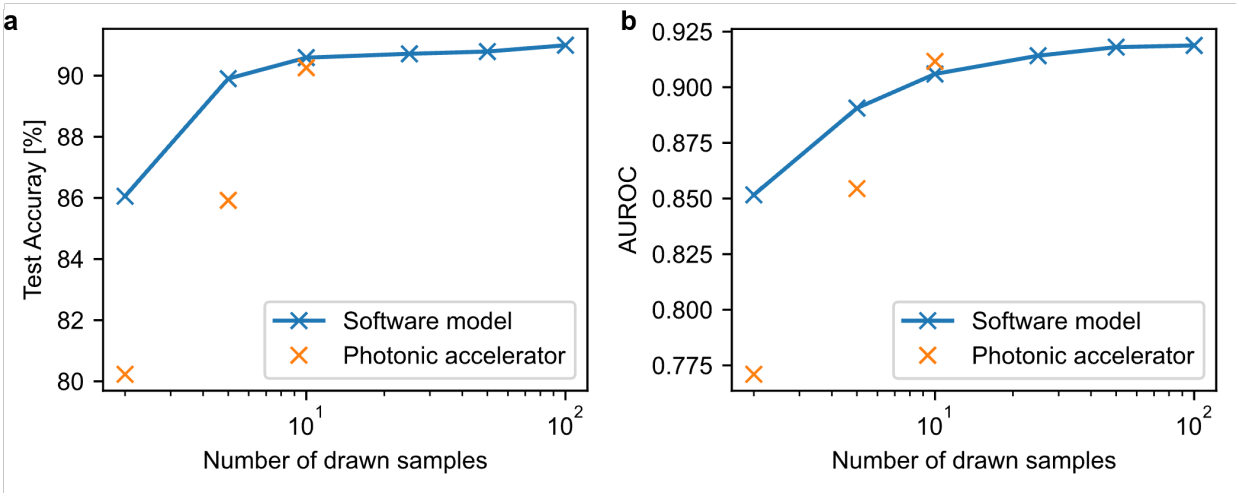

Fig. S6: Comparison between the software model and photonic accelerator. a,b, In the software model, the MedMNIST classification accuracy (**a**) and AUROC (**b**) saturate after approximately ten predictive samples. We therefore use ten predictive samples for experimental inference, as additional samples provide only marginal improvements. At this sample count, the hardware results agree closely with the software predictions.

Table S1: Comparison of Full and Partial Dataset Inference. The software model is only slightly disturbed by running on only 1/5th of the data. In fact, some metrics remain unaffected. From the close match between software and hardware we infer that the subsampling has negligible effect on the final hardware results.

Model	**ID Accuracy [%]**	**AUROC f-MNIST [%]**	**AUROC a-MNIST [%]**
Photonic Accelerator	96.01	84.4	88.0
Software (1/5 data)	96.91	85.6	89.6
Software (full data)	97.01	85.6	89.3