Sampling Parallelism for Fast and Efficient Bayesian Learning

Asena Karolin Özdemir
Karlsruhe Institute of Technology
Karlsruhe, Germany
asena.oezdemir@kit.edu

Lars H. Heyen
Karlsruhe Institute of Technology
Karlsruhe, Germany
lars.heyen@kit.edu

Arvid Weyrauch
Karlsruhe Institute of Technology
Karlsruhe, Germany
arvid.weyrauch@kit.edu

Achim Streit
Karlsruhe Institute of Technology
Karlsruhe, Germany
achim.streit@kit.edu

Markus Götz
Helmholtz AI
Karlsruhe Institute of Technology
Karlsruhe, Germany
markus.goetz@kit.edu

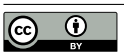
Charlotte Debus
Karlsruhe Institute of Technology
Karlsruhe, Germany
charlotte.debus@kit.edu

Abstract

Machine learning models, and deep neural networks in particular, are increasingly deployed in risk-sensitive domains such as healthcare, environmental forecasting, and finance, where reliable quantification of predictive uncertainty is essential. However, many uncertainty quantification (UQ) methods remain difficult to apply due to their substantial computational cost. Sampling-based Bayesian learning approaches, such as Bayesian neural networks (BNNs), are particularly expensive since drawing and evaluating multiple parameter samples rapidly exhausts memory and compute resources. These constraints have limited the accessibility and exploration of Bayesian techniques thus far. To address these challenges, we introduce sampling parallelism, a simple yet powerful parallelization strategy that targets the primary bottleneck of sampling-based Bayesian learning: the samples themselves. By distributing sample evaluations across multiple GPUs, our method reduces memory pressure and training time without requiring architectural changes or extensive hyperparameter tuning. We detail the methodology and evaluate its performance on a few example tasks and architectures, comparing against distributed data parallelism (DDP) as a baseline. We further demonstrate that sampling parallelism is complementary to existing strategies by implementing a hybrid approach that combines sample and data parallelism. Our experiments show near-perfect scaling when the sample number is scaled proportionally to the computational resources, confirming that sample evaluations parallelize cleanly. Although DDP achieves better raw speedups under scaling with constant workload, sampling parallelism has a notable advantage: by applying independent stochastic augmentations to the same batch on each GPU, it increases augmentation diversity and thus reduces the number of epochs required for convergence.

CCS Concepts

• **Computing methodologies** → **Parallel algorithms**; *Neural networks*; **Knowledge representation and reasoning**.



PASC '26, Bern, Switzerland

ACM ISBN 979-8-4007-2734-4/2026/06
https://doi.org/10.1145/3815572.3815745

Keywords

Parallel Computing, Uncertainty Quantification, Neural Networks

ACM Reference Format:
Asena Karolin Özdemir, Lars H. Heyen, Arvid Weyrauch, Achim Streit, Markus Götz, and Charlotte Debus. 2026. Sampling Parallelism for Fast and Efficient Bayesian Learning. In *Platform for Advanced Scientific Computing Conference (PASC '26), June 29-July 01, 2026, Bern, Switzerland*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3815572.3815745

1 Introduction

Reliable uncertainty estimates for neural network predictions support the transparency, accuracy, and trustworthiness requirements that are anticipated to gain substantial importance under the EU AI Act [40]. As such, scalable and efficient methods for uncertainty quantification (UQ) are essential to provide such estimates, especially in high-risk domains such as atmospheric forecasting [25], healthcare [1], and finance [29], where critical decisions directly depend on model outputs.

Numerous approaches to quantify different aspects of uncertainty have been proposed, and recent trends have demonstrated the benefit of probabilistic loss functions or the utilization of diffusion models to quantify uncertainty [26, 33]. However, probabilistic treatment of a neural network itself, especially in a fully Bayesian framework, is still lacking, as the high computational cost associated with training these models remains a key barrier. Most existing methods rely on stochastic sampling from probability distributions, which is expensive both in time and memory. The sampling process not only slows down training but also requires storing multiple sampled model instances, often turning into a performance bottleneck, or even becoming infeasible when the required samples exceed available GPU memory [2] .

Despite the growing demand, practical implementation of state-of-the-art *sampling-based Bayesian learning* in neural networks is oftentimes restricted to small proof-of-concept models and applications [11]. However, modern networks now routinely contain billions of parameters across domains such as molecular modeling [19, 22], climate and weather prediction [3, 23], and large-scale language modeling [6, 17], making sampling-based Bayesian learning computationally infeasible. Research on practical and scalable approaches for sampling-based Bayesian learning remains limited, even though there are many use-cases in which having uncertainty estimates would be beneficial.

To address these challenges, we explore sampling parallelism, a strategy designed to improve runtime and memory efficiency in uncertainty-aware neural network training using sampling-based Bayesian learning. By reducing the computational barriers associated with sampling, sampling parallelism enables broader experimentation and deployment of these methods. In an exemplary experimental evaluation, we demonstrate that sampling parallelism provides a complementary scaling axis to existing parallelization approaches for neural networks and can aid in pushing scalability limits while even improving model convergence.

2 Background

Uncertainty in neural networks can have two sources: the inherent randomness of the world (aleatoric, data driven), and the lack of knowledge (epistemic, model driven) [20], both of which need to be quantified to navigate risk-sensitive applications of machine learning. There are a variety of methods that enable UQ, however, they oftentimes require sampling which can be computationally expensive as well as memory intensive [10, 24, 28, 31]. In this paper, we discuss Bayesian neural networks (BNNs) with mean-field variational inference, and Monte Carlo dropout (MCD) as examples, however the introduced parallelization concept can be applied to other sampling-based methods such as ensemble models [24], Markov Chain Monte Carlo (MCMC) algorithms [7] or sampling in General Adversarial Networks (GANs) [12] without loss of generality.

2.1 Bayesian Neural Networks and Variational Inference

In BNNs the goal is to learn the probability distribution over all possible outputs, given the inputs and the training data. Mathematically, a BNN is expressed as

$$p(y^* \mid x^*, \mathcal{D}) = \int p(y^* \mid x^*, \mathbf{w})\, p(\mathbf{w} \mid \mathcal{D})\, d\mathbf{w}$$

where $\mathcal{D}$ is the training data, $\mathbf{w}$ are the model weights, x^* is the input and y^* is the output.

However, the computation of the so-called *posterior* $p(\mathbf{w} \mid \mathcal{D})$ is generally intractable. To circumvent this, the method of *Variational Inference (VI)* [4, 14] approximates the posterior by optimizing a parametrized distribution $q(\mathbf{w} \mid \theta)$ such that $q(\mathbf{w} \mid \theta) \approx p(\mathbf{w} \mid \mathcal{D})$. The most common choice for $q(\mathbf{w} \mid \theta)$ is a Gaussian distribution where the parameters θ are the mean μ and the standard deviation σ, i.e. each model weight is described by a μ and σ value.

With this approximation, the loss function of the optimization problem in the Bayesian model formulation above can be reformulated to maximizing the so-called Evidence Lower Bound (ELBO):

$$\mathcal{L}(\theta) = \mathbb{E}_{q(\mathbf{w}|\theta)}\left[\log p(\mathcal{D} \mid \mathbf{w})\right] - \mathrm{KL}(q(\mathbf{w} \mid \theta) \,\|\, p(\mathbf{w}))$$

where the first term $\mathbb{E}_{q(\mathbf{w}|\theta)}\left[\log p(\mathcal{D} \mid \mathbf{w})\right]$ is the data fitting term, which describes how well the current posterior approximation $q(\mathbf{w} \mid \theta)$ fits the data. The second term $-\mathrm{KL}(q(\mathbf{w} \mid \theta) \,\|\, p(\mathbf{w}))$ is the prior matching term. $p(\mathbf{w})$ represents the assumptions that are made about the posterior before any data has been taken into account. Since this distribution captures prior assumptions, it is referred to as the *prior*. The prior matching term draws the current posterior $q(\mathbf{w} \mid \theta)$ towards the prior, by minimizing the Kullback-Leibler (KL) divergence, which is a measure that indicates distinguishability between two probability distributions.

Practically, training a BNN with VI is performed as follows: First, the parameter values θ of the prior $q(\mathbf{w} \mid \theta)$ are initialized. Choosing a Gaussian prior, the means of the weight distribution are initialized in the same fashion as for a non-Bayesian network, while the standard deviations are initialized to a constant value that depends on the layer size. For each epoch and each batch, n sets of parameters (random samples) are drawn from the current weight distributions. For each of these sampled model weights, a forward pass is performed to obtain a model prediction. The distribution over these predictions from all sampled weights is aggregated to obtain an averaged prediction as well as an uncertainty in terms of a standard-deviation of the predictions. The ELBO (rescaled with the dataset size to avoid overflow) is calculated and the backwards pass is performed to update the parameters of the weight distributions. Algorithm 1 illustrates the algorithmic flow of the described procedure.

Algorithm 1 Bayesian Neural Network Training with Variational Inference

Require: Dataset $\mathcal{D}$, number of epochs E, number of samples S

Initialize variational parameters μ and σ for each weight
for epoch = 1 to E **do**
  for each minibatch $(x, y) \in \mathcal{D}$ **do**
    for s = 1 to S **do**
      Sample $\epsilon_s \sim \mathcal{N}(0, I)$
      $w_s \leftarrow \mu + \sigma \odot \epsilon_s$ // reparameterization trick
      $\hat{y}_s \leftarrow \mathrm{ForwardPass}(x, w_s)$
    end for
    $\mathcal{L}_{\text{data}} \leftarrow \frac{1}{S} \sum_{s=1}^{S} \mathrm{Loss}(\hat{y}_s, y)$
    $\mathcal{L}_{\text{KL}} \leftarrow \frac{1}{2} \sum_i \left(\sigma_i^2 + \mu_i^2 - 1 - \log \sigma_i^2\right)$ // Prior Matching
    $\mathcal{L}_s \leftarrow \mathcal{L}_{\text{data}} + \frac{1}{|\mathcal{D}|} \mathcal{L}_{\text{KL}}$ // negative scaled ELBO
    Compute gradients $\nabla \mathcal{L}(\mu, \sigma)$
    Update μ and σ using optimizer (e.g., Adam)
  end for
end for

2.2 Monte Carlo Dropout

BNNs are very powerful tools for UQ; however, they tend to be difficult to implement and train to sufficient accuracy, especially given the high computational demand and memory footprint. A simplified approach that is often used is Monte Carlo Dropout (MCD), which can be interpreted as a variational approximation to the posterior over the network weights, effectively multiplying each weight by a Bernoulli distribution.

Normally, dropout is used as a regularization technique to prevent overfitting in neural networks [38]. It randomly sets a subset of neurons to zero during training, effectively training an ensemble of sub-networks. At inference time, dropout is disabled to use the model at full capacity. Gal and Ghahramani [10] demonstrated that enabling dropout at inference time leads to a form of approximate Bayesian inference. Multiple forward passes with different dropout masks produce a distribution of predictions, where the mean serves

as the model's output and the standard deviation captures epistemic uncertainty. The number of forward passes, s, determines the fidelity of the predictive distribution: larger s give more accurate uncertainty estimates but increase inference time.

While MCD does not lead to a full Bayesian neural network, it offers a practical and easy-to-implement approximation without modifying the model architecture. However, the requirement of performing multiple forward passes for a single prediction slows down inference, particularly for large models. Thus, MCD suffers from similar problems as full BNNs regarding computational complexity and memory demands of samples.

3 Related Works

The general approach to deal with time and memory constraints in computational problems is to parallelize and distribute the problem to multiple processes. For neural network training, parallelization can be performed along different axes.

3.1 Distributed Data Parallelism

In many cases, the primary obstacle to fast training is the need to process large, complex datasets efficiently. Modern neural network training often involves iterating over millions of data points in a sequential batched manner, leading to a time-consuming process. As a result, a practical strategy for accelerating training is to distribute the work associated with each batch, including data loading as well as the forward and backward passes, across multiple GPUs. By partitioning the dataset into distinct shards and assigning them to different compute units, the training of separate portions of the data can proceed in parallel, reducing overall training time and improving hardware utilization.

This approach is referred to as Distributed Data Parallelism (DDP) and is a widely used strategy for scaling the training of deep neural networks across multiple accelerator devices, i.e., GPUs. Since only the data is parallelized in DDP, each GPU still holds a full replica of the model. After each GPU loads its share of the data on which it performs the forward and backward pass, the locally calculated gradients are synchronized across all GPUs in an all-reduce operation to update the model instances on all GPUs. This approach allows for balanced workload distribution, and advanced strategies to efficiently overlap computation and communication, leading to near-linear speedup in most cases. The existence of optimized off-the-shelf frameworks such as torch-distributed [27, 32] has led to a wide acceptance and implementation of the method. Even though communication can become a bottleneck at scale, DDP is still considered the most effective method to speed up training.

One issue that arises when using DDP at scale is that the effective batch size increases linearly with the number of GPUs, since each GPU handles a mini-batch independently. While larger batch sizes can stabilize training, they can also reduce overall model accuracy and generalization due to convergence to sharper minima [13, 21]. This effect is known as *large batch effects*. To circumvent large batch effects and maintain convergence, the learning rate and optimization schedule can be adjusted. However, such tuning is often based on heuristics, that are not always effective in every application.

Another challenge in DDP is the fact that the entire model, including all parameters, intermediate activations, and optimizer states, needs to fit into the memory of a single GPU. This results in a non-negligible memory demand, which becomes even more pressing in the Bayesian setting, where multiple samples of the model are drawn from the parameter distribution. While DDP provides the advantage that more batches can be processed in parallel, it becomes thus less suitable for scalable sampling-based Bayesian learning, when multiple samples of very large models need to be stored in GPU memory.

3.2 Model Parallelism

With growing model sizes, the memory demand associated with holding all model parameters in GPU memory becomes increasingly challenging. In response, model parallelism (MP) has emerged as a way to distribute model parameters across GPUs and, in some cases, even parallelize the computation. Different ways to split the model exist, with each of them coming with its own benefits as well as caveats [5].

In *tensor parallelism*, each layer within a neural network is split across multiple GPUs, and the mathematical operations within each layer are performed in parallel. This enables the model to train even when individual layers are too large to fit into memory. Tensor parallelism can be effective when communication between GPUs is quite fast. However, given the complex operations and layer types that are used in modern neural networks, tensor parallelism presents with substantial communication overheads, and requires careful orchestration and dedicated fine-tuning on very fast interconnects to be efficient. A simpler alternative is *pipeline parallelism*, in which networks are distributed across layers, i.e., individual layers or groups thereof are distributed across GPUs [18, 30]. However, while pipeline parallelism is easy to implement and can efficiently address memory bottlenecks for large models, the sequential nature of the forward-backward-pass computation prohibits true parallel execution.

As neural network architectures advance, so do the strategies to distribute their computational load. In particular, the rise of Transformer-based architectures has pushed the field, given the massive size of these models and the memory demand associated with the attention mechanism. The most common approach among them is MegatronLM [37], a parallelization strategy specifically taylored to Transformer blocks, which has been proven to work efficiently and is available as off-the-shelf library for widespread use.

3.3 Sharded Parallelism

Some methods don't fit into either the data parallelism nor model parallelism category, because they share some properties with both. One of them is sharded parallelism, which distributes a model's parameters, gradients, and optimizer states across GPUs. This greatly reduces memory usage while preserving the standard data-parallel compute pattern: each GPU still performs the full forward and backward pass on its own batch. Before executing each layer, the required parameter shards are temporarily gathered, used for computation, and then released so that only local shards remain. This makes sharded parallelism both data-parallel in terms of computation and model-parallel in terms of storage [41].

Compared to standard Distributed Data Parallel (DDP), which fully replicates the model on every device, sharded parallelism enables training larger models by avoiding redundant memory copies. Fully sharded Data Parallel (FSDP) implements this approach by performing layer-by-layer gather–compute–shard cycles. Although this increases communication relative to plain DDP, it provides substantial memory efficiency and scalability benefits for large models.

3.4 Parallelization of Uncertainty Quantification and Sampling-Based Methods

While other parallelization techniques that we discussed previously can be applied to uncertainty quantification methods as well, there are also parallelization approaches that specifically target UQ methods. Deep ensembles are inherently parallelizable, as independent models can be trained concurrently across multiple devices without the need for communication in between training steps, yielding both predictive performance and well-calibrated uncertainty estimates [24]. Diffusion models use sampling in their denoising steps [16], which are usually sequential and time-consuming since many of them are required just to process a single data item. Recent work demonstrates that these sequential steps can be parallelized for better performance [36]. However, to the best of our knowledge, sampling parallelism in the context of BNNs and MCD has not been explored so far. While it may seem trivially parallel at first, there are several potential pitfalls, for example, the exact vs. approximate communication of the sample averages, which we elaborate on below. Moreover, the combination with data augmentation towards faster model convergence marks another innovation of our approach, which so far has not been explored.

4 Methodology

Sampling-based UQ methods, and BNNs in particular, are inherently expensive to train, and pose substantial challenges with respect to computational and memory demand. For one, they comprise more parameters than their deterministic counterparts, since each weight is replaced by a distribution that is characterized by at least two parameters instead of one, making them by a factor larger. Moreover, drawing s parameter samples during a forward pass effectively scales the model size by a factor of s, further augmenting the memory footprint. In addition, generating these samples and performing separate forward and backward passes for each of them substantially increases computational cost.

These factors have hindered the implementation of large-scale UQ methods in practice thus far. The goal of our work is to target these challenges through parallelization and by that improve the accessibility of research on BNNs and related UQ methodologies beyond toy examples.

4.1 Sampling Parallelism

Our approach targets the reduction of the additional per-GPU memory demands and the parallelization of the computational burden by distributing precisely what makes sampling-based UQ methods more complex and memory demanding: the samples. Figure 1 illustrates the underlying idea.

In our approach, the s random parameter samples are distributed to the p GPUs that are available for training. Thus, each GPU is responsible for drawing s/p samples and performing the forward pass on them. This reduces the memory requirements of model training by a factor of up to p as well as speeding up the training.

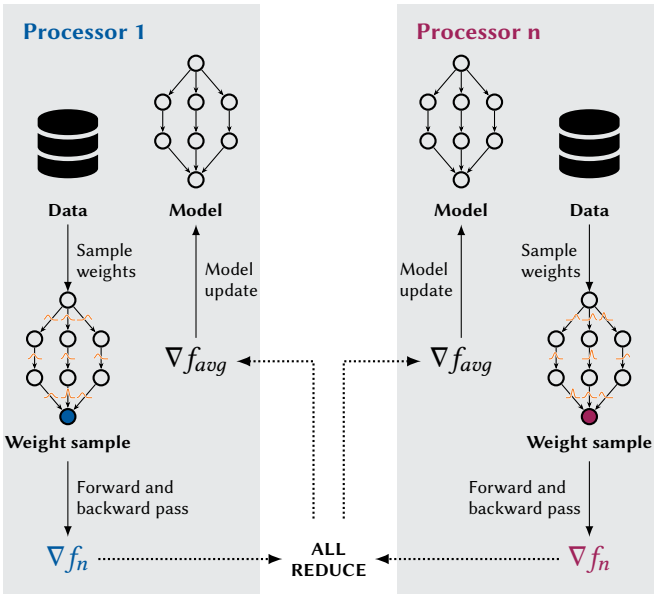


Figure 1: BNN Training with Parallel Sampling

Algorithm 2 illustrates the implementation of sampling parallelism on the example of BNNs trained with VI, and highlights the differences to conventional BNN VI training (c.f. 1). Each GPU loads the same mini-batch from the dataset to ensure consistent data input across computations. While this replication introduces a significant overhead that initially limits scalability, it provides a distinct advantage: each duplicated batch can undergo separate random data augmentations, thereby enhancing augmentation diversity.

Sampling parallelism enables multiple stochastic forward passes to be performed on an identical model–dataset pair while employing distinct random seeds. These seeds govern not only the sampling of network parameters, but also the stochastic components of the data pipeline, including random data augmentations. Extensive prior work has demonstrated that increased diversity and frequency of data augmentation improve generalization performance [5]. Consequently, sampling parallelism effectively yields additional augmented training instances without incurring extra sequential training cost. Formally, each parallel run corresponds to a joint Monte Carlo draw from the distribution over model parameters and stochastic transformations.

Because BNNs represent model weights as probability distributions rather than point estimates, every forward pass requires drawing samples from these weight distributions. To properly capture their variability, multiple weight samples are drawn at every iteration during both training and inference. Accordingly, each GPU samples model weights using a unique random seed, ensuring that no redundant samples are generated. Each GPU then performs

a forward pass to compute its local predictions. The ELBO loss for the current batch is evaluated independently on each GPU, and the corresponding gradients are computed locally. Finally, an all-reduce operation is performed to average and synchronize the gradients across GPUs, after which the model parameters are updated. This synchronization ensures that all GPUs remain consistent and that the updates reflect contributions from multiple independent weight samples, thereby reducing gradient variance and stabilizing training.

Algorithm 2 Sampling Parallel Bayesian Neural Network Training with Variational Inference

Require: Dataset $\mathcal{D}$, number of epochs E, number of samples S

Initialize variational parameters μ and σ for each weight
for epoch = 1 to E **do**
  for each minibatch $(x, y) \in \mathcal{D}$ **do**
    for each GPU p in parallel **do**
      for s = 1 to S/P **do**
        Sample $\epsilon_s \sim \mathcal{N}(0, I)$
        $w_{p,s} \leftarrow \mu + \sigma \odot \epsilon_s$
        $\hat{y}_{p,s} \leftarrow \text{ForwardPass}(x, w_{p,s})$
        $\mathcal{L}_{\text{p,s,data}} \leftarrow \text{Loss}(\hat{y}_{p,s}, y)$
        $\mathcal{L}_{\text{KL}} \leftarrow \frac{1}{2} \sum_i \left(\sigma_i^2 + \mu_i^2 - 1 - \log \sigma_i^2\right)$
        $\mathcal{L}_{p,s} \leftarrow \mathcal{L}_{\text{p,s,data}} + \frac{1}{|\mathcal{D}|} \mathcal{L}_{\text{KL}}$
        Compute gradients $\nabla \mathcal{L}_s(\mu, \sigma)$
      end for
      All-reduce communication from all GPUs
      Average gradients $\frac{1}{S} \sum_{p=1}^{P} \sum_{s=1}^{S/p} \nabla \mathcal{L}_{p,s}(\mu, \sigma)$
      Update μ and σ using optimizer (e.g., Adam)
    end for
  end for
end for

It is important to note that the distributed implementation is not an exact replica of the non-distributed algorithm. While the prior matching part of the ELBO loss can be calculated and aggregated without additional approximations by simply averaging the gradients between GPUs, the data term in general can depend on non-linear functions of, e.g., mean and standard deviation of the samples. However, when these samples are distributed across multiple GPUs, obtaining these statistics becomes nontrivial. In particular, standard deviations cannot be correctly aggregated through simple averaging; thus, relying solely on a gradient averaging yields only an approximation.

A simple example would be the case of model trained on a classification task with a cross-entropy loss on the (arithmetic) mean of the class probabilities of the individual predictions. In the extreme case of the s samples being distributed onto $p = s$ GPUs, each with a single prediction, the effective loss would correspond to aggregating the class probabilities with a geometric mean instead.

Despite this discrepancy, the distributed implementation with only gradient averaging remains an approximation of the local algorithm, and our empirical results indicate that it exhibits very similar behavior in practice. This approach allows us to rely on existing, highly optimized frameworks that handle gradient communication and synchronization automatically.

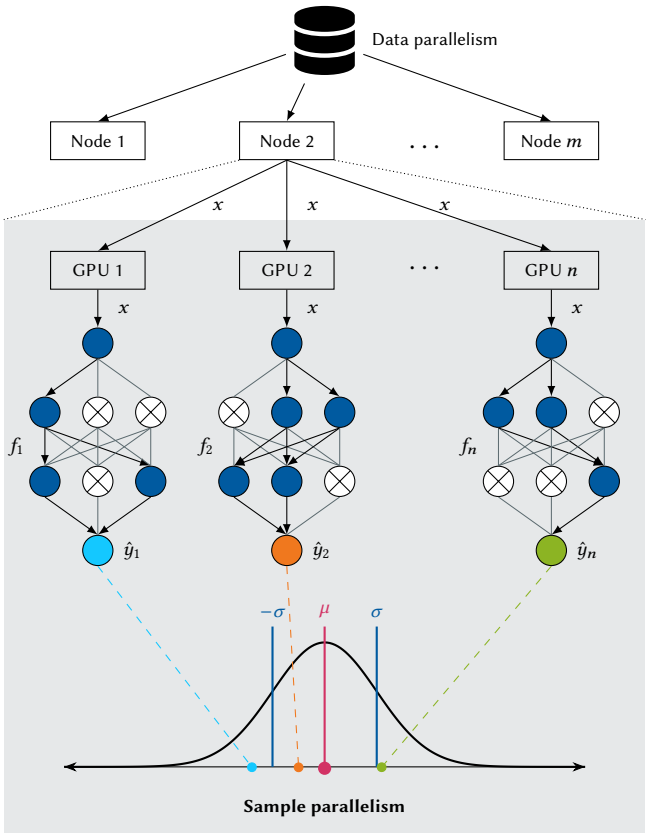


Figure 2: Distributing Training with Hybrid Parallelization

Nonetheless, an exact algorithm where the standard deviation and mean are aggregated across multiple GPUs is also possible. This requires the additional communication of two parameters.

4.2 Combining Sampling Parallelism with DDP

Sampling parallelism aims to tackle the root cause of computation and memory demand specifically introduced by the sampling-based nature of Bayesian learning techniques. As such, it adds an additional axis for parallelization that can be leveraged orthogonally to existing approaches. We illustrate this, by combining our baseline sampling parallelism algorithm with Distributed Data Parallelism (DDP) in a hybrid parallelization strategy. In this setup, weight samples are distributed across GPUs within the same node, while the data are distributed across multiple nodes. This design allows us to leverage the strengths of both approaches. Figure 2 shows how parallelization is handled in this hybrid parallelization.

5 Experimental Setup

We evaluate the proposed algorithms and demonstrate their benefits, feasibility, and scalability using three different use cases. For clarity, we focus primarily on BNNs trained with VI, though the

underlying concepts and methods readily extend to other sampling-based Bayesian learning approaches, datasets, architectures, and tasks. We demonstrate this by extending the experiments to a use case using MCD.

5.1 Hardware and Software

All experiments were performed on a high-performance computing cluster, where each node features 4 NVIDIA 40GB A100 GPUs connected via NVLink3, and 2 AMD EPYC 7402 CPUs with 512 GB RAM, managed through Slurm.

All experiments were conducted using `Python 3.12.3` and `CUDA 12.8`. We employed `PyTorch 2.9.0` [32] and `torch_blue` [34] as the primary frameworks for Bayesian learning, and used `torch.distributed` together with `DistributedDataParallel` [27] for parallelization.

5.2 Use Case 1: ViT on CIFAR-10

The primary use case we evaluate our approach on is the task of image classification. We employ a Bayesian version of a Vision Transformer (ViT) architecture [8], and train it with sampling-parallel variational inference. The ViT architecture features 4×4 patches and an embedding dimension of 192 with three attention heads (64 dimensions per head). The Transformer encoder contains six layers, each followed by an MLP block with a hidden size of 768 (a 4× expansion). To model parameter uncertainty, all weights are assigned a mean-field normal variational distribution with a corresponding mean-field normal prior. Parameter means are initialized using Kaiming initialization and variances are initialized using a constant initialization scaled inversely with the layer width. The variational parameters are optimized via standard VI with Bayes-by-Backprop [4], maximizing the ELBO with a categorical predictive distribution for classification. The ViT model outputs one set of class probabilities for each sampling of its weights.

As dataset, we use **CIFAR-10** consisting of 60,000 images (32×32 pixels each), with the canonical 50k/10k train/test split, while reserving 10% of the training set for validation. We apply a standard set of data augmentations: Each image is randomly cropped to 32×32 with a padding of 4 pixels, then horizontally flipped with a probability of 50%. Images are converted to tensors and normalized using the CIFAR10 mean and standard deviation for each channel. During validation and testing, the predictive performance of the ViT is assessed using top-1 classification accuracy computed from the model's predictive mean.

5.3 Use Case 2: Time-Series Forecasting with an MLP

To illustrate the generalizability of our approach towards all sampling-based Bayesian learning methods, we further explore its application to MCD. For this use case we chose the task of time-series forecasting, using a simple Multilayer Perceptron (MLP) with two hidden layers, each featuring a width of 128 neurons. The choice of a comparatively small model allows us to further study the speedup and efficiency of sampling parallelism in scenarios in which the GPUs are not fully utilized.

We train the Bayesian MLP using the **ENTSO-E de** dataset [9], which contains electricity consumption data for Germany in 15 min intervals over 5 years. The forecasting task is to predict the electricity consumption of the next 6 hours (24 data points) given the consumption of the previous 24 hours (96 data points). The MLP outputs a set of scalar predictions for each random sample and each of the 24 target time points and is trained on the Mean Squared Error (MSE) loss of the averaged predictions. We also use the MSE of the average predictions for the purposes of validation and testing.

5.4 Use-Case 3: Bayesian Weather Models

While use cases one and two allow us to distinctly study performance and scaling behavior of our sampling parallelism approach, they are ultimately too small to fully highlight its unique advantages. In particular, for the Bayesian ViT, the increased computational load and memory demand can be tackled using simple DDP: By decreasing the local batch size per GPU, more memory becomes available for the model and its samples without decreasing efficiency and GPU utilization. While this approach is technically limited by the minimum local batch size of one, the comparatively small size of the CIFAR-10 data items makes this limit practically impossible to reach.

However, this is not the case for applications where individual data points themselves are very large. A paradigm in this regard is data-driven weather forecasting, where data items consist of high-resolution global maps of atmospheric state variables. The field is currently heavily researched and the immense sizes of data and models alike are pushing the boundaries of what is capable with modern accelerator hardware. Already now, state-of-the-art networks can often only be trained using both model and data parallel approaches, with the latter operating under the constraint that only one or two data items fit into accelerator memory. Moreover, these models tend to exhibit large batch effects rather quickly (around an effective global batch size of 8 to 16, according to our experience), which inherently limits the scalability of DDP. This makes the application of sampling-based Bayesian learning methods in these models, in particular BNN versions thereof, virtually impossible.

To illustrate how our approach of sampling parallelism can aid in overcoming this challenge, we examine this application in a third use case. The intention of this evaluation is to function primarily as a representative for large regression models that are typically too big to make Bayesian, rather than as a model tailored for this specific task. Using sampling parallelism, we demonstrate that sampling-based Bayesian learning can be applied to models of this scale.

We use a simple Shifted Window (SWIN) Transformer architecture based on To et al. [39] without the learned positional embedding for the task of weather forecasting. The patch size is set to 2×2 and the embedding dimension to 540. We use 6 SWIN blocks with 12 heads each before and after down- and upsampling and 4 SWIN blocks with 24 heads each in-between down- and upsampling. Figure 3 shows the architecture of this model. It has over 200 million parameters, and each data item is roughly 17.7 MBs.

To model parameter uncertainty, all weights of the model are assigned a mean-field normal variational distribution with a corresponding mean-field normal prior. The network is initialized using Kaiming initialization for the means and a constant initialization scaled inversely with the layer width for the variances. The

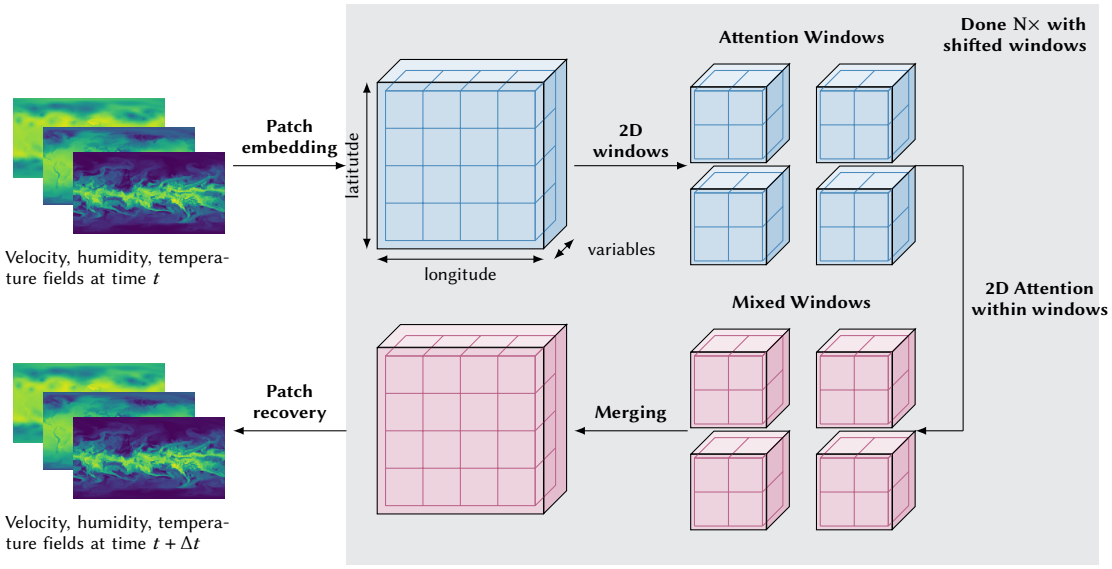


Figure 3: Simplified illustration of the SWIN transformer architecture used for the weather forecasting task; patch embedding and recovery is performed via a 2d convolution or transpose convolution layer with kernel size = stride = $(2, 2)$ over the latitude and longitude dimensions

variational parameters are optimized via standard VI with Bayes-by-Backprop [4]. We train the model on the **ERA5** dataset [15] at 1.5° resolution (downloaded via Weatherbench 2 cloud storage [35]), restricted to three-hour subsampling (i.e., only 00:00 UTC, 03:00 UTC, etc.) and data from 1980 to 2020. The task is a 6h forecast of the same 69 variables used in the Pangu-Weather model [3]. We also add the same constant masks as part of the input as well as longitude and latitude features combined with time-of-day and day-of-year information respectively, resulting in data samples with dimensions $76 \times 121 \times 240$. The data is augmented by random periodic shifts in longitude. The weather SWIN Transformer outputs a set of forecasts for each variable and grid point. We train the model by maximizing the ELBO, using a Gaussian negative log-likelihood for the data term in ELBO. Validation and testing are performed via latitude-weighted RMSEs of individual variables. However, since predictive performance was not the focus of this, these models were not trained to convergence and thus the RMSEs are not reported.

The source code and evaluation scripts used in our experiments are available open-source[1].

6 Results & Discussions

To evaluate the feasibility and efficiency of our proposed parallelization scheme, we perform a series of scaling experiments in which we observed both the runtime as well as the convergence of accuracy values. We conduct two types of experiments. For one, we explore scaling experiments in which the workload is scaled proportionally to the increasing resources to establish that our parallelization scheme works reasonably well. This type of scaling experiment, which we will be calling proportional-sample scaling, evaluates how runtime changes when both the problem size and the number of GPUs grow proportionally, keeping the workload per GPU constant. In our case the workload is determined by the number of samples drawn. *Efficiency* is then evaluated as $E(p) = \frac{T_1(n)}{T_p(n \cdot p)}$. In our proportional-sample scaling experiments, we start with a non-Bayesian model on a single GPU and increase the number of samples drawn proportional to the number of GPUs, keeping every other hyperparameter fixed.

Proportional-sample scaling highlights how an algorithm behaves when parallel workload grows to exclude the diminishing returns associated with fixed-size problems. However, ultimately, the goal of parallelization is to accelerate computation of a fixed workload by adding additional compute resources, i.e., fixed-sample scaling. Fixed-sample scaling measures how the runtime of a fixed-size problem improves as the number of processing units increases. Its primary evaluation metric is *speedup*, defined as the ratio of the single-GPU runtime to the runtime using p GPUs $S(p) = \frac{T_1(n)}{T_p(n)}$, where $T_1(n)$ is the required time for a task of size n on a single GPU and $T_p(n)$ is the corresponding cost when the same task is distributed onto p GPUs. This metric characterizes how effectively additional compute resources reduce computation time and reveal upper limits imposed by the serial portion of the workload.

For our fixed-sample scaling experiments, we increase the number of GPUs while keeping the workload (number of random samples) constant. There are two variations on how we conduct these experiments. In one setup, we do not change any other hyperparameter. While this enables the model to maintain the same setup, it effectively reduces the load per GPU making memory and time usage inefficient. In the other case, we ensure GPU usage at maximum capacity by increasing the batch size proportionally. For comparative reasons, we also parallelize the same model via DDP. Again we study two cases, to match our experiments with sampling parallelism. For DDP, the equivalent for case 1 is, to keep global batch size constant, meaning the batch size per GPU decreases. For

[1] https://github.com/RAI-SCC/sampling-parallelism

the second case, we keep the local batch size constant, ensuring that GPUs run at capacity. As for sampling and hybrid parallelism, we also investigate the two variants that maintain either the global batch size or the GPU load.

6.1 ViT on CIFAR-10

We start by evaluating the efficiency of sampling parallelism by conducting proportional-sample scaling on the task of image classification, using a Bayesian ViT on CIFAR-10. In a non-distributed setting, an epoch with 16 samples takes about 1.5 minutes for this task.

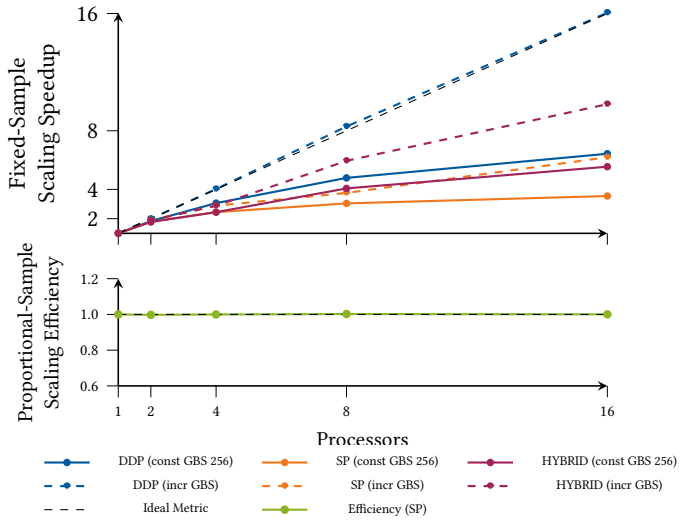


Figure 4: Speedup (fixed-sample scaling, top) and efficiency (proportional-sample scaling, bottom) of training the Bayesian ViT on CIFAR10. For fixed-sample scaling, we compare different sampling parallelism, data parallelism, and a hybrid of both, using 16 random samples and either a global batch size of 256 or an increasing global batch size of 256 times number of GPUs. For proportional-sample scaling we use a fixed global batch size of 1024 and scale the number of random samples from 1 to 16.

The results depicted in Figure 4 (bottom) show that sampling parallelism achieves near ideal scaling behavior; efficiency remains effectively constant over an increasing number of GPUs, with only minor point-to-point fluctuations attributable to measurement noise rather than algorithmic overhead. This near-perfect scaling indicates that the sampling procedure parallelizes cleanly, with no observable degradation in performance as the number of samples (and GPUs) increases. This is unsurprising since the task of distributed sampling in the way we propose it is embarrassingly parallel. Consequently, the method is well-suited for large-scale sampling-based uncertainty estimation approaches in which many independent samples must be evaluated in parallel.

Another key point is that increasing the number of drawn random samples from 1 to 16 would not have been feasible without the proposed parallelization scheme, due to memory limitations, unless the batch size was altered as well. While alternative strategies, such as parallelizing other components of the model (tensor parallelism, pipelining etc.) to free up additional memory and compute time, could in principle enable larger sample counts, these approaches are considerably more intrusive and difficult to apply in practice. In contrast, our method provides a straightforward and scalable mechanism for distributing sample evaluations across multiple GPUs, enabling substantial increases in sampling throughput with minimal changes to the underlying model.

For the fixed-sample scaling experiments, we fixed the total number of samples to 16. As noted previously, this configuration cannot be executed on a single GPU without modification, so we reduced the local batch size to at most 256 to ensure that the experiment remained feasible under the same hardware constraints. All other settings were kept identical to the proportional-sample scaling setup, as well as between the different methods.

We then conduct an inter-parallelism comparison between data parallelism (DDP), sampling parallelism, and the hybrid approach combining both (see Section 4.2). The hybrid parallelization uses sampling parallelism for intra-node distribution and DDP for inter-node. For each strategy, we conducted the experiment in two variants: one in which the global batch size is held constant across GPU count, meaning that the local batch size and therefore load per GPU decreases, and one in which the global batch size is increased so that each device operates at full capacity. The achieved speedups based on the recorded per epoch run-times are summarized in Figure 4. When operating at capacity, DDP achieves near-ideal speedup, whereas hybrid and sample-parallel configurations fall short of perfect scaling. This gap arises because, in sampling parallelism, the same data batch must be loaded on every GPU, whereas DDP distributes different microbatches across devices. Since data loading constitutes a significant portion of the runtime, the duplicated data loading in sampling parallelism naturally limits the achievable speedup. However, as outlined before, the duplicated data loading in sampling parallelism allows us to apply independent stochastic data augmentations locally per sample, thus increasing the augmentation diversity. By loading the same data batch on all GPUs, we can apply different random augmentations, such as random crops and flips, on each of our GPUs for the same batch. When examining the corresponding convergence of the validation accuracy (c.f. Figure 5), we find that sampling parallelism converges substantially faster than DDP when per epoch accuracy increases are considered.

We investigate this effect further by analyzing the convergence behavior of sampling parallelism under two augmentation settings: one in which all GPUs apply identical augmentations, and one in which each GPU applies its own stochastic augmentations. The results depicted in Figure 6 show that convergence is noticeably faster when GPUs perform independent augmentations. These findings strengthen the usefulness of sampling parallelism. By increasing the variation of augmentations, sampling parallelism can effectively enlarge the dataset seen during training, enabling faster convergence and better generalizability. While this effect could in principle also be replicated on the individual minibatch of a single GPU in DDP, this would require replicating the loaded data before augmentation, incurring additional compute and memory costs compared to sampling parallelism.

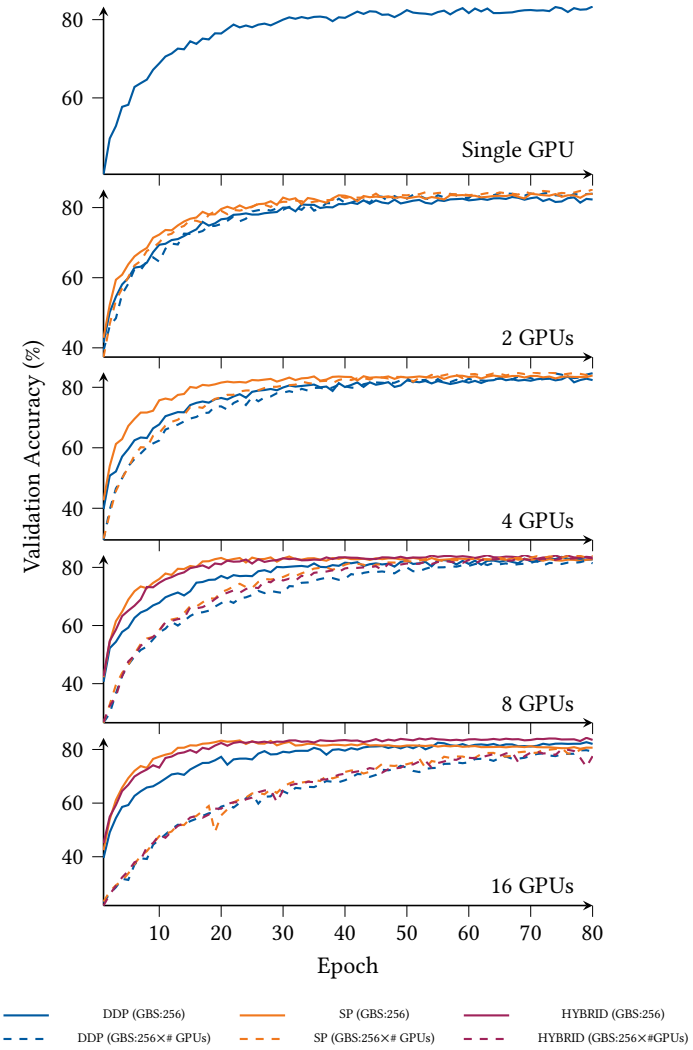


Figure 5: Accuracy of the Bayesian ViT on CIFAR10 with two different global batch sizes and 16 random samples, using sampling parallelism, DDP and hybrid parallelism, over the course of training, for different numbers of GPUs.

Although DDP is more efficient in terms of epoch time, sampling parallelism requires fewer epochs to reach comparable accuracy. To assess overall convergence speed, we therefore compare accuracy as a function of wall-clock time, shown in Figure 7. When viewed in this manner, the total time to reach a target accuracy is similar across all methods, with sampling parallelism showing a slight advantage due to its accelerated convergence in terms of epochs.

Taken together, these fixed-sample scaling results highlight an important trade-off between computational efficiency and convergence behavior. While DDP is the most effective approach for minimizing per-epoch runtime and achieves the best raw speedup under full device utilization, sampling parallelism's ability to leverage augmentation diversity across GPUs enables significantly faster learning per epoch, which compensates for its weaker scaling properties.

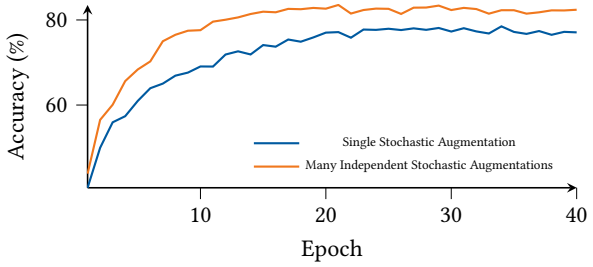


Figure 6: Validation accuracy of sampling parallelism over the course of training, using 16 random samples on 16 GPUS and two data augmentation settings: one in which all GPUs apply identical augmentations (blue), and one in which each GPU applies its own stochastic augmentations (orange).

Consequently, despite its higher data-loading overhead, sampling parallelism can match, and in some cases surpass, DDP in terms of wall-clock time to reach a target accuracy.

To evaluate the impact not only on the model predictive performance, but also the quality of the uncertainty quantification, we examine the negative log likelihood (NLL) and the mean absolute calibration error as measures for uncertainty. Figure 8 shows the progression of the NLL over the epochs of training with different global batch sizes as well as different parallelization strategies. While NLL decreases for all configurations (see Figure 8), the sample parallel implementation yields better performance, due to the benefit of enhanced data augmentation. Table 1 shows the mean absolute calibration error, which is a measure of how well predicted uncertainty levels match the empirical errors of the model. Sampling parallelism yields consistently lower MACE, compared to DDP.

Table 1: Mean Absolute Calibration Error (MACE ↓) of the ViT on CIFAR-10 trained on 16 GPUs with 16 samples given parallelization strategy and global batch size.

Method/GBS	256	512	1024	2048
DDP	0.1407	0.1399	0.1753	0.1525
SP	0.1211	0.1121	0.0657	0.0534

6.2 MLP on ENTSO-E de

We further study the generalizability of the approach to different sampling-based Bayesian learning methods by transitioning from BNNs with VI to MCD, using the task of time series forecasting with an MLP. In a non-distributed setting, an epoch with 16 samples takes about 10 seconds for this task. Although sampling parallelism remains fully functional for the MLP in the sense that training can be carried out without instability, predictive performance matches expectations, and the communication patterns operate smoothly, its efficiency is considerably limited.

Figure 9 (bottom) shows the efficiency from proportional-sample scaling experiments. We observe that efficiency begins to decline sharply beyond four GPUs. This behavior is expected, as the communication overhead associated with sampling parallelism grows with the number of GPUs and quickly becomes significant compared

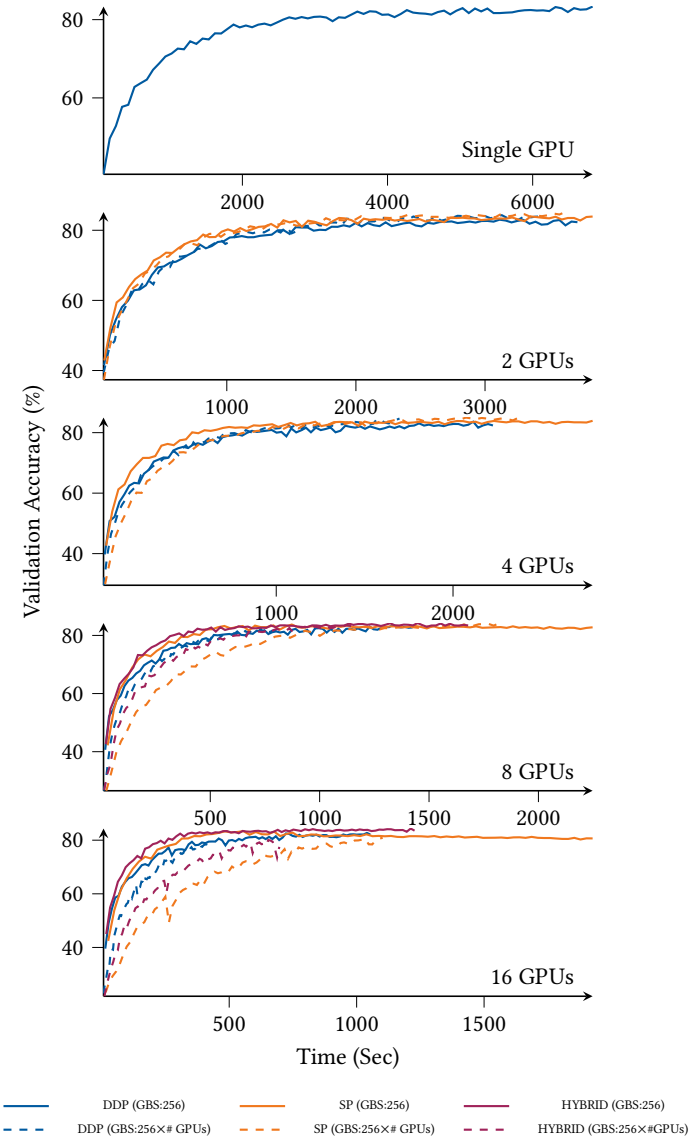


Figure 7: Accuracy of the Bayesian ViT on CIFAR10 with two different global batch sizes and 16 random samples, using sampling parallelism, DDP, and hybrid parallelism, with respect to wall-clock time, for different numbers of GPUs.

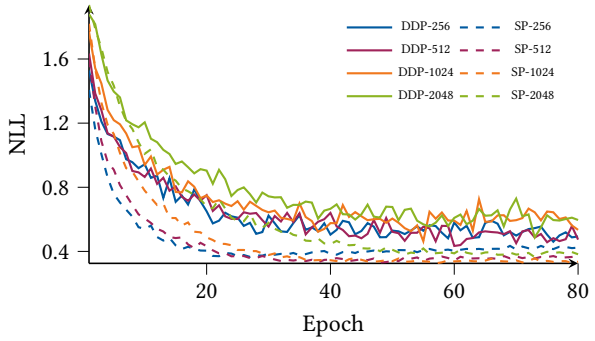


Figure 8: Negative log likelihood during training of the ViT on CIFAR-10 trained on 16 GPUs with 16 samples labeled with parallelization strategy and global batch size.

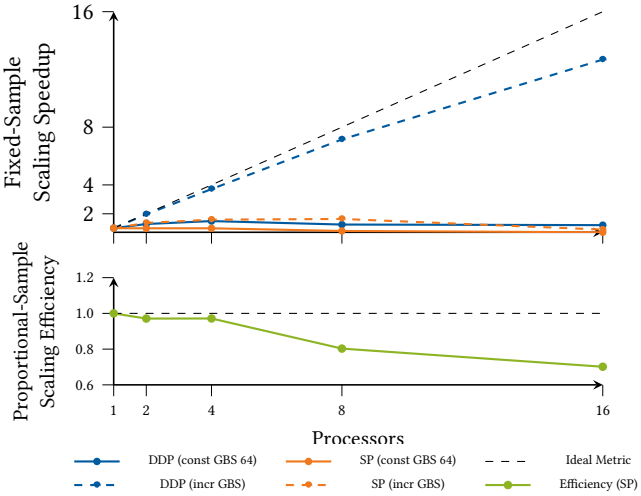


Figure 9: Speed up (fixed-sample scaling, top) and efficiency (proportional-sample scaling, bottom) of MCD on an MLP with the ENTSO-E dataset. For fixed-sample scaling, we compare sampling parallelism and data parallelism, using 16 random samples and either a global batch size of 64 or an increasing global batch size of 64 times number of GPUs. For proportional-sample scaling we use a fixed global batch size of 1024 and scale the number of random samples from 1 to 16.

to the relatively low computational cost of evaluating this small model. In other words, for small networks, the cost of synchronizing gradients and managing multiple weight samples can outweigh the benefits of parallel computation. This highlights that the advantages of sampling parallelism are most pronounced for larger models, where the computational workload per GPU is substantial enough to amortize the communication overhead and achieve high scaling efficiency. This effect becomes even more pronounced in fixed-sample scaling experiments (c.f. Figure 9 top), where the workload per GPU decreases as more resources are added, eventually causing communication overhead to dominate and speedup to drop sharply. In contrast, DDP performs much better in these scenarios, as it parallelizes the data loading process, a step that cannot be fully vectorized on a single GPU, allowing for more effective utilization of additional GPUs.

6.3 SWIN transformer on ERA5

The previous experiments have demonstrated the general feasibility and efficiency of sampling parallelism, and the potential to even improve model convergence via data augmentation. However, the

raw speedup achieved through sampling parallelism is ultimately not competitive with DDP in those cases.

To demonstrate that by opening up an additional parallelization axis through sampling parallelism, which allows us to tackle the unprecedented challenges in large-scale Bayesian neural networks, we conduct proportional-sample scaling experiments on a SWIN Transformer for weather prediction using the ERA5 data. Performing fixed-sample scaling would require comparing results obtained on multiple GPUs to those from a single GPU running the same workload. Due to hardware limitations, it is not possible to accommodate the full workload on a single GPU. This limitation underscores the practical necessity of sampling parallelism for very large models. In a non-distributed setting, an epoch with 2 samples takes about 40 hours for this task.

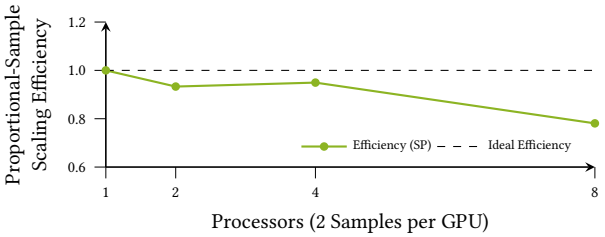


Figure 10: Scaling efficiency of the Bayesian SWIN Transformer with the ERA5 dataset, where the number of random samples equals 2 times the number of GPUs, and the local batch size per GPU is 1.

Given that the used loss requires at least two random samples per GPU, the only configuration that fit into the GPU memory was a local batch size of one and exactly two random samples per GPU. Our proportional-sample scaling experiments depicted in Figure 10 demonstrate the efficiency of scaling on up to 8 GPUs and 2 random samples per GPU, which enables training on 16 samples, which would not have been possible without parallelizing the samples. The proportional-sample scaling efficiency is at a reasonable level, remaining above 90% for up to four GPUs and dropping to 80% for eight. Regardless of the scaling behavior, the fact that training on this many samples becomes possible at all shows, how important it is to be able to distribute these samples as this method allows training beyond the boundaries of DDP alone.

The presented results demonstrate that sampling parallelism is a valuable technique for sampling-based uncertainty quantification, particularly in settings where data augmentation plays a critical role in predictive performance. Moreover, sampling parallelism can be flexibly integrated with other parallelization strategies, as illustrated by our hybrid approach, making it a versatile tool that can be tailored to the specific requirements of a given task. However, we do not advise to use sampling parallelism in very small networks, as the scalability is sub-par.

7 Conclusion

With this work, we aim to tackle the significant computational burden associated with training sampling-based methods, in particular BNNs, thereby facilitating their widespread practical adoption. Since this burden originates primarily from the need to evaluate multiple samples of the model parameter distributions, we introduce sampling parallelism as a strategy that distributes random sample evaluations across multiple processes. This approach makes large-scale Bayesian learning in neural networks substantially more tractable with respect to both memory requirements and runtime. While duplicating data-loading operations across GPUs introduces overhead and can hinder raw speedup in comparison to other parallelization techniques such as DDP, it also enables each GPU to apply independent stochastic augmentations to the same batch. This increases the diversity of the training data, which can improve convergence and generalization. Thus, the method presents not only a computational trade-off but also a methodological opportunity. A shortcoming of sampling parallelism is that each loss function that is used needs to be carefully examined to determine if simple gradient synchronization between GPUs is sufficient for exact replication of the sequential algorithm. If not, either further communication will be required or the parallelization will be an approximate replica whose effectiveness will need to be determined via experiments. As future work, developing a dedicated package that implements and documents loss functions known to preserve exactness under such parallelization schemes would be highly beneficial, as it would reduce the burden on practitioners and help standardize reliable usage. Nonetheless, our experiments show that the training behavior is not negatively affected in some use-cases.

Sampling parallelism complements existing parallelization techniques, such as data or model parallelism, by providing an additional axis of scalability. Our hybrid experiments illustrate that sampling parallelism can be combined effectively with other methods, allowing practitioners to tailor their parallelization strategy to the computational structure of their task.

Finally, sampling parallelism becomes especially valuable when both models and datasets grow large. When individual data samples or model components already saturate the memory of a single GPU (such as in atmospheric modeling), other parallelization techniques become a necessity. In these cases, sampling parallelism provides a crucial additional degree of freedom for parallelizing and distributing a network, enabling uncertainty-aware learning.

While our experiments focus on BNNs and Monte Carlo Dropout, the underlying approach is applicable to a much broader family of Bayesian and UQ techniques. Evaluating performance across additional methods, architectures, tasks, and datasets will further map out its generality and practical scope, and allow us to position sampling parallelism as a practical and versatile tool for large-scale Bayesian modeling in real-world applications.

Acknowledgments

This work is supported by the German Federal Ministry of Research, Technology and Space (BMFTR) under the 01IS22068 EQUIPE grant and the 01LK2313A SMARTWEATHER21 grant, and by the Helmholtz AI platform grant. The authors gratefully acknowledge the computing time made available to them through the HAICORE@KIT partition and on high-performance computer HoreKa at the NHR Center KIT via the SmartWeather21-p0021348 NHR large project. This work was further supported by the Helmholtz Association's Initiative and Networking Fund on the HAICORE@FZJ partition.

References

[1] Abdullah A Abdullah, Masoud M Hassan, and Yaseen T Mustafa. 2022. A review on bayesian deep learning in healthcare: Applications and challenges. *IEEe Access* 10 (2022), 36538–36562. doi:10.1109/ACCESS.2022.3163384

[2] Daniel Andrade and Koki Sato. 2025. On the effectiveness of partially deterministic Bayesian neural networks. *Computational Statistics* 40, 5 (2025), 2491–2518. doi:10.1007/s00180-024-01561-7

[3] Kaifeng Bi, Lingxi Xie, Hengheng Zhang, Xin Chen, Xiaotao Gu, and Qi Tian. 2023. Accurate medium-range global weather forecasting with 3D neural networks. *Nature* 619, 7970 (2023), 533–538. doi:10.1038/s41586-023-06185-3

[4] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. 2015. Weight Uncertainty in Neural Network. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 37)*, Francis Bach and David Blei (Eds.). PMLR, Lille, France, 1613–1622. https://proceedings.mlr.press/v37/blundell15.html

[5] Felix Brakel, Uraz Odyurt, and Ana-Lucia Varbanescu. 2024. Model parallelism on distributed infrastructure: A literature review from theory to LLM case-studies. arXiv:2403.03699 [cs.DC]

[6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[7] Arkabandhu Chowdhury and Christopher Jermaine. 2018. Parallel and distributed MCMC via shepherding distributions. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 1819–1827.

[8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv:2010.11929 [cs.CV] doi:10.48550/arXiv.2010.11929

[9] ENTSO-E. 2025. Germany – Load Data, Transparency Platform. https://transparency.entsoe.eu/.

[10] Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 48)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.). PMLR, New York, New York, USA, 1050–1059. https://proceedings.mlr.press/v48/gal16.html

[11] Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseok Lee, Matthias Humt, Jianxiang Feng, Anna Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, et al. 2023. A survey of uncertainty in deep neural networks. *Artificial Intelligence Review* 56 (2023), 1513–1589. doi:10.1007/s10462-023-10562-9

[12] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2020. Generative adversarial networks. *Commun. ACM* 63, 11 (2020), 139–144.

[13] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2018. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. arXiv:1706.02677 [cs.CV] doi:10.48550/arXiv.1706.02677

[14] Alex Graves. 2011. Practical variational inference for neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems* (Granada, Spain) *(NIPS'11)*. Curran Associates Inc., Red Hook, NY, USA, 2348–2356.

[15] Hans Hersbach, Bill Bell, Paul Berrisford, Shoji Hirahara, András Horányi, Joaquín Muñoz-Sabater, Julien Nicolas, Carole Peubey, Raluca Radu, Dinand Schepers, et al. 2020. The ERA5 global reanalysis. *Quarterly journal of the royal meteorological society* 146, 730 (2020), 1999–2049. doi:10.1002/qj.3803

[16] Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising diffusion probabilistic models. *Advances in neural information processing systems* 33 (2020), 6840–6851.

[17] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack W. Rae, and Laurent Sifre. 2022. Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, Article 2176, 15 pages.

[18] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. 2019. GPipe: efficient training of giant neural networks using pipeline parallelism. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, Article 10, 10 pages.

[19] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. 2021. Highly accurate protein structure prediction with AlphaFold. *nature* 596, 7873 (2021), 583–589. doi:10.1038/s41586-021-03819-2

[20] Alex Kendall and Yarin Gal. 2017. What uncertainties do we need in Bayesian deep learning for computer vision?. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 5580–5590.

[21] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2017. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. arXiv:1609.04836 [cs.LG] doi:10.48550/arXiv.1609.04836

[22] Rohith Krishna, Jue Wang, Woody Ahern, Pascal Sturmfels, Preetham Venkatesh, Indrek Kalvet, Gyu Rie Lee, Felix S Morey-Burrows, Ivan Anishchenko, Ian R Humphreys, et al. 2024. Generalized biomolecular modeling and design with RoseTTAFold All-Atom. *Science* 384, 6693 (2024), eadl2528. doi:10.1126/science.adl2528

[23] Thorsten Kurth, Shashank Subramanian, Peter Harrington, Jaideep Pathak, Morteza Mardani, David Hall, Andrea Miele, Karthik Kashinath, and Anima Anandkumar. 2023. FourCastNet: Accelerating Global High-Resolution Weather Forecasting Using Adaptive Fourier Neural Operators. In *Proceedings of the Platform for Advanced Scientific Computing Conference* (Davos, Switzerland) *(PASC '23)*. Association for Computing Machinery, New York, NY, USA, Article 13, 11 pages. doi:10.1145/3592979.3593412

[24] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. 2017. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 6405–6416.

[25] Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Ferran Alet, Suman Ravuri, Timo Ewalds, Zach Eaton-Rosen, Weihua Hu, et al. 2023. Learning skillful medium-range global weather forecasting. *Science* 382, 6677 (2023), 1416–1421. doi:10.1126/science.adi2336

[26] Christian Lessig, Ilaria Luise, Bing Gong, Michael Langguth, Scarlet Stadtler, and Martin Schultz. 2023. AtmoRep: A stochastic model of atmosphere dynamics using large scale representation learning. arXiv:2308.13280 [physics.ao-ph] doi:10.48550/arXiv.2308.13280

[27] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. 2020. PyTorch distributed: experiences on accelerating data parallel training. *Proc. VLDB Endow.* 13, 12 (Aug. 2020), 3005–3018. doi:10.14778/3415478.3415530

[28] Wesley J. Maddox, Timur Garipov, Pavel Izmailov, Dmitry Vetrov, and Andrew Gordon Wilson. 2019. A simple baseline for Bayesian uncertainty in deep learning. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, Article 1179, 12 pages.

[29] Akib Mashrur, Wei Luo, Nayyar A Zaidi, and Antonio Robles-Kelly. 2020. Machine learning for financial risk management: a survey. *Ieee Access* 8 (2020), 203203–203223. doi:10.1109/ACCESS.2020.3036322

[30] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. 2019. PipeDream: generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (Huntsville, Ontario, Canada) *(SOSP '19)*. Association for Computing Machinery, New York, NY, USA, 1–15. doi:10.1145/3341301.3359646

[31] Radford M Neal. 2012. *Bayesian learning for neural networks*. Vol. 118. Springer Science & Business Media, Heidelberg, Germany. doi:10.1007/978-1-4612-0745-0

[32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: an imperative style, high-performance deep learning library. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, Article 721, 12 pages.

[33] Ilan Price, Alvaro Sanchez-Gonzalez, Ferran Alet, Tom R Andersson, Andrew El-Kadi, Dominic Masters, Timo Ewalds, Jacklynn Stott, Shakir Mohamed, Peter Battaglia, et al. 2025. Probabilistic weather forecasting with machine learning. *Nature* 637, 8044 (2025), 84–90. doi:10.1038/s41586-024-08252-9

[34] RAI-SCC. 2025. torch_blue. https://github.com/RAI-SCC/torch_blue.

[35] Stephan Rasp, Stephan Hoyer, Alexander Merose, Ian Langmore, Peter Battaglia, Tyler Russell, Alvaro Sanchez-Gonzalez, Vivian Yang, Rob Carver, Shreya Agrawal, et al. 2024. WeatherBench 2: A benchmark for the next generation of data-driven global weather models. *Journal of Advances in Modeling Earth Systems* 16, 6 (2024), e2023MS004019.

[36] Andy Shih, Suneel Belkhale, Stefano Ermon, Dorsa Sadigh, and Nima Anari. 2023. Parallel sampling of diffusion models. *Advances in Neural Information Processing Systems* 36 (2023), 4263–4276.

[37] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. arXiv:1909.08053 [cs.CL] doi:10.48550/arXiv.1909.08053

[38] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15, 56 (2014), 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html

[39] Deifilia To, Julian Quinting, Gholam Ali Hoshyaripour, Markus Götz, Achim Streit, and Charlotte Debus. 2024. Architectural insights into and training methodology optimization of Pangu-Weather. *Geoscientific Model Development* 17, 23 (2024), 8873–8884. doi:10.5194/gmd-17-8873-2024

[40] Matias Valdenegro-Toro and Radina Stoykova. 2024. The Dilemma of Uncertainty Estimation for General Purpose AI in the EU AI Act. arXiv:2408.11249 [cs.AI] doi:10.48550/arXiv.2408.11249

[41] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. 2023. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel. arXiv:2304.11277 [cs.DC] doi:10.48550/arXiv.2304.11277

Received 12 December 2025; revised 20 March 2026