URSABENCH: A SYSTEM FOR COMPREHENSIVE BENCHMARKING OF BAYESIAN DEEP NEURAL NETWORK MODELS AND INFERENCE METHODS

Meet P. Vadera [1] **Jinyang Li** [*2] **Adam D. Cobb** [*3] **Brian Jalaian** [4] **Tarek Abdelzaher** [2] **Benjamin M. Marlin** [1]

ABSTRACT

While deep learning methods continue to improve in predictive accuracy on a wide range of application domains, significant issues remain with other aspects of their performance, including their ability to quantify uncertainty and their robustness. Recent advances in approximate Bayesian inference hold significant promise for addressing these concerns, but the computational scalability of these methods can be problematic when applied to large-scale models. In this paper, we present *URSABench* (the Uncertainty, Robustness, Scalability, and Accuracy Benchmark), an open-source suite of models, inference methods, tasks and benchmarking tools. URSABench supports comprehensive assessment of Bayesian deep learning models and approximate Bayesian inference methods, with a focus on classification tasks performed both on server and edge GPUs.

1 INTRODUCTION

As deep learning models continue to improve in terms of predictive accuracy in many application domains, significant issues remain in terms of their performance with respect to other highly important aspects of performance, including their ability to robustly quantify uncertainty (Guo et al., 2017) and provide sensible output in the presence of out of distribution examples (Ovadia et al., 2019).

Approximate Bayesian inference methods (Neal, 1996; Jaakkola & Jordan, 2000) hold considerable promise for addressing such issues with deep learning models. In the past, the computation and storage requirements of traditional approximate Bayesian inference methods have made them significantly more expensive to apply when compared to traditional stochastic gradient descent-based learning methods (Bottou, 2010). However, recent advances have significantly improved the feasibility of deploying approximate Bayesian inference methods to increasingly larger deep learning models (Welling & Teh, 2011; Louizos & Welling, 2017; Zhang et al., 2020). While highly promising, much of this prior research has lacked a focus on comprehensive evaluation including the simultaneous assessment of multiple aspects of performance (e.g., in-distribution accuracy, out-of-distribution detection, adversarial robustness, uncertainty calibration, storage costs, computational costs, etc.). There has also been a lack of systematic comparisons across different methods, including accounting for optimizing hyperparameters of inference methods. As a result, there is a significant information gap around which methods hold the most promise in different regions of the multi-faceted trade-off space for different down-stream tasks.

This paper advances the area of approximate Bayesian inference for deep learning models by addressing the question of how to fairly and efficiently perform comprehensive evaluations of methods. We propose a suite of benchmarking tools and artifacts including hyperparameter optimization methods, benchmark models, benchmark inference methods, benchmark datasets, and benchmark tasks. We call this benchmarking system *URSABench*, the Uncertainty, Robustness, Scalability, and Accuracy Benchmark.

We provide a set of benchmark results obtained using the URSABench system that compare a wide array of recent methods across multiple tasks and evaluation metrics. We also provide the first comprehensive view of the scalability of different approximate Bayesian deep learning approaches on edge hardware through the use of prediction run-time benchmarking on the Jetson TX2 platform using TensorRT implementations of fit models.

The organization of the paper is as follows. In Section 2 we present background and related work on Bayesian supervised learning and approximate Bayesian inference methods with a focus on methods for approximate Bayesian deep learning. In Section 3 we discuss principles for evaluating Bayesian deep learning models and algorithms including predictive performance, calibration, robustness, and scalabil-

[*]Equal contribution [1]University of Massachusetts Amherst [2]University of Illinois Urbana-Champaign [3]SRI International [4]US Army Research Laboratory. Correspondence to: Meet P. Vadera <mvadera@cs.umass.edu>, Adam D. Cobb <adam.cobb@sri.com>.

Proceedings of the 5^{th} MLSys Conference, Santa Clara, CA, USA, 2022.

ity. In Section 4 we present the URSABench benchmarking system components that encapsulate the evaluation principles described in Section 3. In Section 5 we present and discuss benchmarking results obtained using the system, including results on edge GPU hardware.

We hope that the open-source benchmarking infrastructure that we present will be a highly useful tool for the research community, and that it will contribute to exposing areas with the greatest performance gaps to help focus future research efforts.[1]

2 Bayesian Supervised Learning and Approximate Bayesian Inference

In supervised learning, the dataset $\mathcal{D}$ consists of labeled instances $\{(\mathbf{x}_i, y_i)|1 \leq i \leq N\}$. $\mathbf{x}_i \in \mathbb{R}^D$ is the feature vector and $y_i \in \mathcal{Y}$ is the prediction target. The set of possible targets $\mathcal{Y}$ varies by prediction task. In this section, we review Bayesian supervised learning, focusing on approximation methods for large-scale problems.

2.1 Bayesian Supervised Learning

A probabilistic supervised learning model provides a conditional probability model of the form $p(y|\mathbf{x}, \theta)$ where $\theta \in \mathbb{R}^K$ are the model parameters. The likelihood of a dataset given values for the parameters is given by $p(\mathcal{D}|\theta)$. The standard assumption that the data are independent and identically distributed leads to the equality $p(\mathcal{D}|\theta) = \prod_{i=1}^{N} p(y|\mathbf{x}, \theta)$ (Neal, 1996). Bayesian inference also requires asserting a prior distribution over the model parameters $p(\theta|\theta^0)$ that may itself depend on a set of prior parameters θ^0 (Neal, 1996).

The first key problem in Bayesian inference is the computation of the posterior distribution over the unknown parameters given a training dataset $\mathcal{D}_{tr}$ (consisting of inputs $\mathcal{D}_{tr}^x$, and labels $\mathcal{D}_{tr}^y$) and the prior. The parameter posterior is obtained via Bayes rule and is shown in Equation (1).

$$p(\theta|\mathcal{D}_{tr}, \theta^0) = \frac{p(\mathcal{D}_{tr}^y|\mathcal{D}_{tr}^x, \theta)p(\theta|\theta^0)}{\int p(\mathcal{D}_{tr}^y|\mathcal{D}_{tr}^x, \theta)p(\theta|\theta^0)d\theta} \tag{1}$$

The term in the denominator of the parameter posterior is referred to as the "evidence" and is computationally intractable for most probabilistic supervised machine learning models, including logistic regression and deep neural networks (Neal, 1996). The intractability of the evidence term generally results in the parameter posterior itself being computationally intractable.

However, for most predictive machine learning tasks, we only require the computation of low-dimensional expectations of the parameter posterior. The posterior predictive distribution, which is necessary and sufficient for making predictions, is shown in Equation (2).

$$p(y|\mathbf{x}, \mathcal{D}_{tr}, \theta^0) = \mathbb{E}_{p(\theta|\mathcal{D}_{tr}, \theta^0)}[p(y|\mathbf{x}, \theta)] \tag{2}$$

2.2 Approximate Bayesian Inference for Supervised Learning

Approximate Bayesian inference methods aim to approximate either the parameter posterior or expectations taken with respect to the parameter posterior. There are many possible approaches. We review Monte Carlo, surrogate density (e.g., variational inference methods), and distillation-based methods in this section.

2.2.1 Monte Carlo Methods

Monte Carlo methods are a classical approach to Bayesian computation that approximate the intractable parameter posterior $p(\theta|\mathcal{D}, \theta^0)$ via a distribution constructed from a finite set of samples $p_{MC}(\theta|\mathcal{D}, \theta^0) = \frac{1}{S}\sum_{s=1}^{S} \delta(\theta, \theta_s)$ where $\delta(\theta, \theta')$ is the Dirac delta function (Smith & Roberts, 1993). The samples θ_s must be drawn from the true posterior $p(\theta|\mathcal{D}, \theta^0)$. The sifting property of the Dirac delta function results in unbiased approximations to posterior expectations that converge at a $1/\sqrt{S}$ rate. Equation (3) provides an example in the case of the posterior predictive distribution, which also highlights the connection between ensemble methods and Monte Carlo methods.

$$\begin{aligned} p(y|\mathbf{x}, \mathcal{D}, \theta^0) &= \mathbb{E}_{p(\theta|\mathcal{D}, \theta^0)}[p(y|\mathbf{x}, \theta)] \\ &\approx \mathbb{E}_{p_{MC}(\theta|\mathcal{D}, \theta^0)}[p(y|\mathbf{x}, \theta)] \\ &= \frac{1}{S}\sum_{s=1}^{S} p(y|\mathbf{x}, \theta_s) \end{aligned} \tag{3}$$

The fundamental problem with drawing samples θ_s from the true posterior $p(\theta|\mathcal{D}, \theta^0)$ is that this operation is typically also computationally intractable. Markov Chain Monte Carlo (MCMC) methods solve this problem by constructing a Markov Chain with the true posterior $p(\theta|\mathcal{D}, \theta^0)$ as its equilibrium distribution. Classical MCMC methods include the Gibbs sampler (Casella & George, 1992) and the Metropolis-Hastings sampler (Chib & Greenberg, 1995). Somewhat more recent methods include the Hamiltonian Monte Carlo sampler (Duane et al., 1987), slice sampling (Neal, 2003), and Riemann manifold sampling methods (Girolami & Calderhead, 2011). Samples are generated sequentially by these methods and are thus autocorrelated, but can be thinned to reduce redundancy.

One of the most significant problems when applying the MCMC methods mentioned above to large-scale deep learning models is that the computation required to draw a single sample depends linearly on the size of the dataset N, making them highly computationally expensive to use relative

[1] Our code is available at: https://github.com/reml-lab/URSABench

to stochastic gradient descent-based (SGD) optimization methods that use mini-batches of size $M \ll N$ (Bottou, 2010).

Recent research has seen the proposal of multiple highly promising methods that can potentially overcome this problem including stochastic gradient Langevin dynamics (SGLD) (Welling & Teh, 2011), stochastic gradient Hamiltonian Monte Carlo (Chen et al., 2014), and their cyclic step size versions as presented in Zhang et al. (2020).

An alternative to sampling in the original parameter space $\mathbb{R}^K$ is to sample in a reduced dimensional space $\mathbb{R}^{K'}$ for $K' < K$ and project back to the full-dimensional space. Izmailov et al. (2019) introduces methods for constructing subspaces using SGD iterates. These methods effectively make a bias-variance trade-off. By restricting the sampler to operate in a subspace of $\mathbb{R}^K$, they introduce bias, but by enabling the underlying Markov chain to mix faster, they can decrease variance.

2.2.2 *Surrogate Density Methods*

Another major family of methods are approaches based on approximating the true posterior density via an analytically tractable surrogate distribution $p_S(\theta|\mathcal{D}, \theta^0, \phi)$ where ϕ are auxiliary parameters of the surrogate distribution (Jordan et al., 1999; Jaakkola & Jordan, 2000; Ghosh et al., 2016; Minka, 2001). A common approximation for unconstrained parameters is the use of a multivariate Gaussian distribution with a diagonal covariance matrix $\mathcal{N}(\theta; \mu, \Sigma)$ (i.e., $\phi = [\mu, \Sigma]$), but other more advanced approximations are possible including normalizing flows (Louizos & Welling, 2017).

The fundamental problems in this approach are selecting a computationally tractable function for measuring the discrepancy between the true posterior $p(\theta|\mathcal{D}, \theta^0)$ and the surrogate posterior $p_S(\theta|\mathcal{D}, \theta^0, \phi)$, and minimizing this function with respect to the parameters ϕ of the surrogate posterior. The most commonly used discrepancy function is the Kullback-Leibler (KL) divergence $\mathrm{KL}(p(\theta)||q(\theta)) = E_{p(\theta)}[\log(p(\theta)/q(\theta))]$ (MacKay, 2003).

The KL divergence is not symmetric in its arguments resulting in two different measures. When the surrogate posterior is used as the first argument, the result is the variational inference (VI) framework (Jordan et al., 1999; Jaakkola & Jordan, 2000). When the surrogate posterior is used as the second argument, the result is the expectation propagation (EP) framework (Minka, 2001). These two extremes can also be interpolated by more generalized divergence measures (Li & Turner, 2016).

Advances in the past decade have led to more scalable methods in this family, including stochastic variational inference (Hoffman et al., 2013), that are able to accommodate large-scale datasets using mini-batch gradients computed over sub-sets of data. MC Dropout is a particularly interesting approach that shows that the application of dropout during SGD-based learning approximates stochastic variational inference with a particular non-Gaussian surrogate posterior distribution (Gal & Ghahramani, 2016). There is also recent work using variational inference combined with rank-1 matrices to parameterize layer weights for neural networks that has shown promising results (Dusenberry et al., 2020). We note that unless the true posterior is close to the surrogate approximating family, all methods in this family result in biased posterior estimates where the bias depends both on the discrepancy function used and the form of the approximating family.

Surrogate density methods still have problems producing approximate posterior predictive distributions at deployment time. In particular, even if $p_S(\theta|\mathcal{D}, \theta^0, \phi)$ is a simple parametric distribution, the expectation $\mathbb{E}_{p_S(\theta|\mathcal{D},\theta^0,\phi)}[p(y|\mathbf{x}, \theta)]$ cannot be computed analytically due to the non-linearity of $p(y|\mathbf{x}, \theta)$. As a result, these methods require samples to be drawn from the surrogate posterior in order to make predictions at test time. These methods trade-off the potential bias in the surrogate parameter posterior with the ability to draw independent samples from the simpler surrogate posterior after the parameters have been estimated.

2.2.3 *Distillation-Based Methods*

The final class of methods that we review are posterior distillation-based methods, including Bayesian Dark Knowledge (BDK) (Balan et al., 2015) and Generalized Posterior Expectation Distillation (GPED) (Vadera et al., 2020). Instead of focusing on the parameter posterior, BDK focuses on directly approximating the posterior predictive distribution. It does this by learning an auxiliary neural network model to further approximate a Monte Carlo approximation to the posterior predictive distribution $\mathbb{E}_{p_{MC}(\theta|\mathcal{D},\theta^0)}[p(y|\mathbf{x}, \theta)]$. Similar ideas have also recently been explored in the case of general ensembles (Malinin et al., 2020).

The goal of these distillation approaches is not to improve over the Monte Carlo approximation, but rather to reduce the computation time required to compute the Monte Carlo average at deployment time, which can be very large when S is large and $p(y|\mathbf{x}, \theta)$ is computationally expensive. The GPED framework further generalizes the BDK framework by distilling arbitrary posterior expectations of the form $\mathbb{E}_{p_{MC}(\theta|\mathcal{D},\theta^0)}[f(y, \mathbf{x}, \theta)]$, which can be useful for approximating other properties of the parameter posterior.

2.3 Discussion

As we have described in this section, approaches to approximate Bayesian inference differ substantially in a number

of respects. In principle, MCMC methods are unbiased, but they rely on convergence of Markov chains to produce samples from the posterior. The computation time required to perform a single iteration using classical methods scales linearly with the dataset size N. However, more recent methods have been able to achieve improved scaling using mini-batches that scale with $M \ll N$. The storage cost of these methods is $O(SK)$ where K is the number of parameters in the base model.

Surrogate density methods have essentially complementary properties. They have a fixed bias determined by the approximating family and discrepancy function used. Classical approaches like the use of diagonal Gaussian approximating families require only $O(2K)$ storage, and standard stochastic gradient learning methods can be used to efficiently learn the surrogate posterior parameters.

Both MCMC and surrogate density models typically require the application of sampling to make predictions at test time. When S samples are used, the computational complexity of making predictions is S times higher than if a single instance of the base model is used. Distillation methods can help with this problem by reducing the test-time complexity of making predictions with a posterior ensemble through the selection of a compact student model architecture.

3 Evaluation Principles

While advances in supervised deep learning methods have focused almost exclusively on accuracy over the last decade (Krizhevsky et al., 2012), there are multiple additional properties of models and inference algorithms that are of great interest, particularly for Bayesian deep learning models and methods that aim to better represent uncertainty. We group these properties into several categories including predictive performance, calibration, robustness, and computational efficiency. We discuss each of these categories in this section, as well as tasks, methods, and metrics to evaluate them.

3.1 Predictive Performance

Predictive performance is by far the most widely considered property of supervised machine learning models. A fundamental question when evaluating probabilistic supervised models is how one evaluates the posterior predictive uncertainty.

In the classification setting, accuracy is the coarsest measure of the predictive performance of a probabilistic model. It simply assesses the number of examples for which the true class has the highest predictive probability. Measures like precision, recall, and the F1 score are all similar to accuracy in that they are based on hard predictions. While these measures have strong interpretability properties, they are not particularly sensitive to posterior uncertainty. In particular, accuracy is not affected by the level of uncertainty so long as the most likely class is the correct prediction. Accuracy thus does not differentiate between mild confidence in incorrect predictions and egregiously high confidence in incorrect predictions. In the regression setting, classical evaluation metrics including mean squared error and mean absolute error have a similar issue. While they are sensitive to the predictive mean, they also ignore predictive uncertainty.

In the classification setting, predictive log likelihood can provide a more refined notion of probabilistic predictive performance by assessing the log probability of the true class. This measure is strongly influenced by confident incorrect predictions. In the regression case, the use of predictive log likelihoods for evaluation can also provide more information about the posterior uncertainty of the predictions. However, one of the main drawbacks of predictive log likelihood is that absolute log likelihood values are not as easily interpretable as absolute accuracy values. Predictive likelihood is often better suited for use in relative comparisons between models or inference methods as a result.

Lastly, in general, predictive performance must be assessed on a test dataset $\mathcal{D}_{te}$. The standard assumption is that this test set is sampled from the same distribution as the training dataset $\mathcal{D}_{tr}$. We will refer to this setting as *in-domain prediction*. Recent work in machine learning has put an increasing focus on robustness to encountering out of domain examples at deployment time. We will return to this issue in Section 3.3.

3.2 Calibration

In the binary classification domain, interpreting probability as relative frequency leads to the idea that the fraction of instances that should be classified as positive among a set of instances predicted to be positive with probability p should be exactly p. A model that satisfies this property for all values of p is said to be *perfectly calibrated*. While early work on neural network models indicated they had better calibration properties than some other types of models (Niculescu-Mizil & Caruana, 2005), more recent evaluations of deep learning models have shown that their calibration properties can be quite poor (Guo et al., 2017).

In the case of approximate Bayesian supervised classification, calibration is estimated by computing the predictive probability $p_i = p(Y_i = 1|\mathbf{x}, \mathcal{D}, \theta^0)$ for each test instance i, binning the predictive probabilities, and computing the accuracy within each bin b. Let N_b be the number of instances in bin b, A_b be the accuracy of instances in bin b, and C_b be the average predictive probability (or confidence) of instances in bin b. Plotting the bin accuracies A_b as a bar chart results in a visualization referred to as a *reliability plot*. We can also compute the calibration error for bin b as $\kappa_b = |A_b - C_b|$. The expected calibration error (ECE)

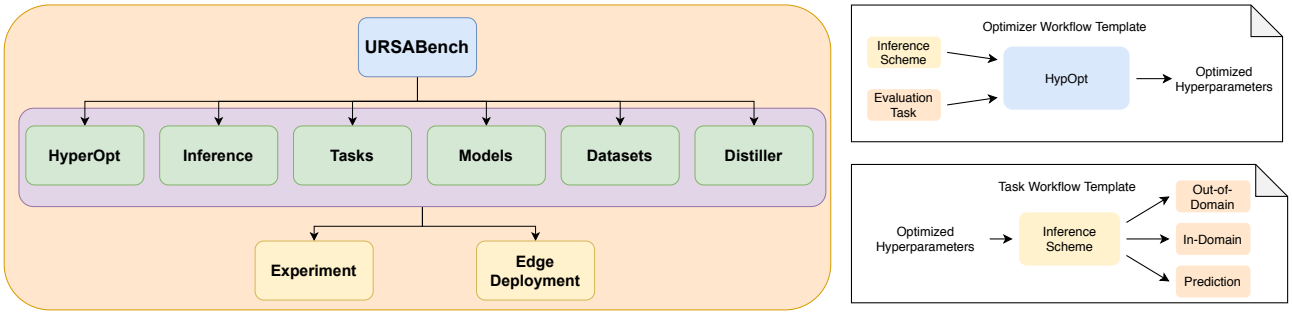

Figure 1. (Left) Individual components of URSABench are shown at the middle level. Based on a configuration of a subset of these components, users can run an end-to-end experiment or deploy the sampled models to an edge environment. (Right, top) A sample workflow template for HyperOpt. Users select an inference scheme and an evaluation task and select the hyperparameters that perform the best for the chosen evaluation scheme. (Right, bottom) A sample workflow template for Inference. Users select hyperparameters (ideally through some form of hyperparameter optimization), and an inference scheme to obtain a Bayesian ensemble. Finally, this Bayesian ensemble is evaluated on different tasks.

is then defined to be $ECE = \frac{1}{N}\sum_b N_b \kappa_b$. The maximum calibration error is given by $MCE = \max_b \kappa_b$ (Guo et al., 2017).

In the case of multi-class classification, we can obtain a generalization of calibration using a one-vs-all formulation. We let $p_{ci} = p(Y_i = c|\mathbf{x}, \mathcal{D}, \theta^0)$ and bin the p_{ci} values separately for each class c. Let N_{cb} be the number of instances in bin b, A_{cb} be the accuracy of instances in bin b, and C_{cb} be the average predictive probability (or confidence) of instances in bin b, all when class c is considered to be the target class. We then define $\kappa_{cb} = |A_{cb} - C_{cb}|$ and $ECE = \frac{1}{NC}\sum_c \sum_b N_{cb}\kappa_{cb}$.

It is important to note that calibration and accuracy are distinct concepts. A model with low accuracy could be perfectly calibrated while a model with high accuracy could be poorly calibrated. Calibration and predictive performance are thus largely orthogonal properties of a model.

The Brier score provides a measure related to the multi-class generalization of expected calibration error (Brier, 1950). Again letting $p_{ci} = p(Y_i = c|\mathbf{x}, \mathcal{D}, \theta^0)$, the multi-class Brier score is given by $BS = \frac{1}{N}\sum_c \sum_i (p_{ci} - [y_i = c])^2$ where $[y_i = c]$ is 1 if $y_i = c$ and is 0 otherwise. The Brier score can be interpreted as mixing together aspects of calibration and predictive performance.

Like prediction performance measures, calibration is assessed on an in-domain test dataset $\mathcal{D}_{te}$ that is assumed to be sampled from the same distribution as the training dataset $\mathcal{D}_{tr}$.

3.3 Robustness

A key aspect of robustness of methods is their ability to provide sensible outputs when input examples are not drawn from the training distribution (Ovadia et al., 2019). Such an evaluation typically leverages a test set that is explicitly out-of-distribution (OOD). A variety of methods have been proposed to leverage the output of a probabilistic model for the purpose of detecting when examples are OOD. Examples include thresholds on the entropy of the posterior predictive distribution (also called the total uncertainty). For approaches that provide access to the posterior distribution over model parameters, methods have been proposed that instead threshold the knowledge uncertainty. Performance is typically measured via OOD prediction as a binary classification task.

Another interesting property of models and inference methods is their robustness to adversarial manipulations (Goodfellow et al., 2015). There are multiple frameworks for generating adversarial examples including the Fast Gradient Sign Method (FGSM) (Goodfellow et al., 2015), Projected Gradient Descent (PGD) (Madry et al., 2018) and the Carlini-Wagner attack (Carlini & Wagner, 2017). The robustness property of interest is the ability of methods to resist adversarial perturbations as measured by the success rate of adversarial attacks. We note that in the Bayesian supervised learning context, these attack methods require access to the approximate posterior predictive distribution function and its gradients, which is a more complex setting than for single point-estimated models. We leave an assessment of adversarial robustness of Bayesian deep learning methods to future work.

3.4 Scalability

We consider two primary aspects to computational scalability: run time and storage cost. As noted in Section 2, different approximate Bayesian inference methods can have widely different properties in terms of both aspects of computational scalability. Of primary interest in this work are how the predictive performance, calibration and robustness

properties of methods trade off against their computational scalability properties. In particular, which approach is "best" depends on what aspects of performance are important for a given task.

For example, the method that gives the best absolute predictive performance on a given task may be acceptable even if it has high run time and storage requirements if the deployment context is a server and there are no real-time constraints on making predictions. However, if there are real-time constraints, a method that is faster may be needed. If the deployment context is an embedded system, methods that are both faster and require reduced storage may be needed.

When benchmarking methods, the storage cost will be estimated via the number of parameters. The run-time of methods can be assessed in different ways, including actual computation time as well as a number of more portable statistics such as the number of floating-point operations (flops) or the number of multiply-accumulate operations (MACs). In this work, we benchmark training run time on server hardware and prediction/inference run time on both server and edge hardware.

4 THE URSABENCH SYSTEM

In this section, we describe the URSABench system, which includes multiple datasets, models, inference methods, and tasks that implement the evaluation principles described in the previous section. URSABench is implemented using the PyTorch deep learning library (Paszke et al., 2019). The current system includes benchmarking components for small-scale, medium-scale, and large-scale models and tasks, as well as evaluation of scalability on server and edge GPU-accelerated platforms. The architecture of URSABench is illustrated in Figure 1. There are six main modules including **HyperOpt**, **Inference**, **Tasks**, **Models**, **Datasets** and **Distiller**. We describe each of them below.

4.1 The HyperOpt Module

For comparisons between methods to be meaningful, hyperparameters of the inference methods and models need to be set for each combination of inference method, model, dataset, and task. The URSABench HyperOpt module provides several hyperparameter optimization approaches including Bayesian optimization, random search, and grid search. The results in the next section use Bayesian optimization as the hyper-parameter optimization method.

4.2 The Inference Module

The Inference module includes implementations of multiple recent inference methods for Bayesian deep learning with a standard interface. Methods in this module allow users to either sample iteratively, or collect all parameter posterior samples at once. This allows greater flexibility in interfacing with downstream tasks or distillation while working under memory constraints. Implemented inference methods include HMC[2], SGLD (Welling & Teh, 2011), SGHMC (Chen et al., 2014), cSGLD, cSGHMC (Zhang et al., 2020), SWAG (Maddox et al., 2019) and PCA-based subspace inference + elliptical slice sampling (PCA+ ESS (SI)) (Izmailov et al., 2019). As baselines, we also provide MC dropout (Gal & Ghahramani, 2016) and SGD-based point estimation.

4.3 The Distiller Module

This module provides methods to distill a posterior ensemble into a student model. Knowledge distillation is useful to reduce the deployment time storage and computation requirements. An implementation of Bayesian Dark Knowledge (BDK) (Balan et al., 2015) based distillation is included. Note that the current implementation distills the posterior back into the starting model architecture as described in the original paper (Balan et al., 2015). As was shown in (Vadera et al., 2020), better performance can often be achieved by distilling into a larger model.

4.4 The Models and Datasets Modules

The small-scale benchmark uses a basic, fully connected MLP with two hidden layers containing 200 units each as the benchmark model, with MNIST providing the benchmark in-domain dataset (LeCun, 1998). At the medium-scale, we use ResNet50 (He et al., 2016) and WideResNet as the benchmark models (Zagoruyko & Komodakis, 2016), with CIFAR10 and CIFAR100 as the benchmark in-domain datasets (Krizhevsky et al., 2009). At the large-scale, we use ResNet50 with ImageNet (Deng et al., 2009) as the benchmark in-domain dataset.

4.5 The Tasks Module

The Tasks module includes a set of inference and prediction tasks using different datasets. The tasks are designed to support assessment of multiple proprieties of models and algorithms, including most of those identified in the discussion of evaluation principles. Specifically, accuracy is assessed using in-domain test sets. To assess uncertainty quantification, we compute negative log likelihood, Brier score, and performance on a misclassification detection task, all using the in-domain test sets. We also consider a decision-making task that focuses on assessing the quality of the tail of the predictive distribution using imbalanced datasets and costs that strongly penalize errors on the rare classes (Cobb et al., 2018) (see Appendix A for details).

[2]HMC is only implemented for tasks where the model and full dataset can fit on the GPU. We use the `hamiltorch` Python package (Cobb et al., 2019).

We assess robustness using an out-of-distribution (OOD) classification task (Ovadia et al., 2019; Vadera et al., 2020) leveraging knowledge uncertainty (see Appendix B for a review of uncertainty decomposition). The small-scale benchmark uses FashionMNIST (Xiao et al., 2017) and KMNIST (Clanuwat et al., 2018) as OOD test sets, while the medium-scale benchmark uses SVHN (Netzer et al., 2011) and STL10 (Coates et al., 2011) as OOD test sets. For the large-scale benchmark, we use ImageNet-Sketch (Wang et al., 2019) as the OOD test set. Performance on OOD tasks is assessed using AUROC. Finally, the benchmark focuses on computation time as the measure of computational scalability, measured in total training time required, including any pre-training and pre-processing that may be required.

4.6 The Run-time Latency Module

We implement a profiling pipeline to evaluate and compare model inference latency. The models optimized by HyperOpt are compiled using TensorRT (NVIDIA Corporation) before deployment. TensorRT is an acceleration library from Nvidia that improves the inference efficiency of already-trained models on Nvidia platforms. At compilation, TensorRT applies several hardware-dependent optimizations to the input model such as quantization, layer fusion, kernel auto-tuning, etc. The compiled model is saved as a TensorRT engine and executed by the TensorRT run-time during the execution phase. This module supports the deployment of Bayesian ensembles and distilled models onto edge hardware for run-time latency benchmarking.

4.7 Experiment Workflows and Composite Scores

Along with the above components, we also provide workflows for executing hyperparameter optimization, inference and evaluation for each of the model-method-dataset-task combination that form the core of the benchmark evaluation process. This includes workflow components for compiling models for edge deployment using TensorRT.

We divide our benchmark into three categories: small-scale, medium-scale, and large-scale. The small-scale benchmark consists of MNIST as its in-domain dataset using MLP models. The medium-scale benchmark consists of all combinations of the CIFAR datasets with the WideResNet28x10 and ResNet50 models. Finally, the large-scale benchmark consists of the ImageNet dataset with the ResNet50 model.

Lastly, the simultaneous assessment of multiple aspects of performance for multiple model, method, dataset and task combinations yields many individual results for each inference method. An important design choice in URSABench is thus to summarize performance in terms of key selected metrics along with composite scores that combine related individual metrics. For accuracy, we include an average over all benchmark models and all in-domain test sets. For robustness, we use an average over all models and OOD datasets. For uncertainty quantification, we separately compute an average over models and datasets in terms of negative log likelihood (NLL) and misclassification task performance.

4.8 Discussion: Extensibility and Reproducibility

The URSABench system is designed to be extensible and reproducible. Researchers who are interested in benchmarking the performance of a new model can add it to the Models module and update the benchmarking workflow template to evaluate the model. Similarly, a researcher interested in benchmarking a new approximate inference algorithm can add it to the Inference module, update the benchmarking workflow template to include the new algorithm, and evaluate its performance. The benchmark predictive performance, robustness and uncertainty evaluations are all fully reproducible based on the existing experimental workflow implementations.

The one aspect of the system that is not fully reproducible is the scalability assessment, as it relies on profiling methods on real hardware. In the prediction run-time results reported in the next section, we use the Jetson TX2 platform. These results are reproducible on that hardware that is available at relatively low cost. The training time results use server GPUs where there is much more variability in available hardware. In this work, we show absolute time for benchmarking training time scalability, but a metric with more potential stability across hardware platforms is the ratio of the absolute training time score for a given method to that of the base SGD method used on the same platform. We leave an exploration of the standardization of training scalability across hardware platforms to future work.

5 URSABench Benchmark Results

In this section, we report benchmark results obtained using the URSABench system. We divide our benchmark results into the three categories mentioned earlier: small-scale, medium-scale and large-scale. Implementation details for the inference schemes are provided in Appendix D.

5.1 Small-Scale Benchmark Results

The small-scale results are displayed in Table 1. The detailed experimental results behind each composite score can be found in Appendix E. The small-scale results show how challenging it can be to distinguish between different approximate inference schemes using relatively simple models and datasets. SGD and SGHMC are both marginally ahead in accuracy and NLL; HMC appears to show the most robust performance in OOD, and SGHMC does best for the uncertainty metric. However, the minor relative difference between all the performance metrics points to focusing on

Table 1. URSABench **small-scale** benchmark performance. (Results presented as mean ± std. dev. across 5 trials.)

Inference	Accuracy ↑	NLL ↓	Robustness ↑	Uncertainty ↑	Training Time ↓
HMC	0.9819 ± 0.0010	0.0593 ± 0.0016	0.9570 ± 0.0075	0.9734 ± 0.0012	178 ± 1.5
SGLD	0.9839 ± 0.0004	0.0492 ± 0.0022	0.9065 ± 0.0377	0.9679 ± 0.0233	765 ± 4.2
SGHMC	0.9862 ± 0.0003	0.0446 ± 0.0003	0.9426 ± 0.0048	0.9807 ± 0.0003	768 ± 2.3
cSGLD	0.9857 ± 0.0003	0.0476 ± 0.0011	0.9521 ± 0.0022	0.9795 ± 0.0007	1063 ± 4.2
cSGHMC	0.9836 ± 0.0009	0.0533 ± 0.0016	0.9276 ± 0.0094	0.9759 ± 0.0015	1111 ± 3.3
PCA + ESS (SI)	0.9840 ± 0.0007	0.0520 ± 0.0016	0.9360 ± 0.0038	0.9695 ± 0.0012	1146 ± 10.2
MC dropout	0.9858 ± 0.0007	0.0501 ± 0.0031	0.9429 ± 0.0059	0.9769 ± 0.0019	377 ± 2.4
SGD	0.9860 ± 0.0002	0.0452 ± 0.0012	-	-	193 ± 0.9

Table 2. URSABench **medium-scale** benchmark performance.

Inference	Accuracy ↑	NLL ↓	Robustness ↑	Uncertainty ↑	Training Time ↓
SGLD	0.863	0.620	0.781	0.908	51117
SGHMC	0.866	0.599	0.777	0.905	51466
cSGLD(50)	0.881	0.453	0.819	0.919	230155
cSGHMC(50)	0.880	0.446	0.812	0.917	231441
cSGLD(9)	0.876	0.484	0.802	0.911	51117
cSGHMC(9)	0.873	0.506	0.802	0.905	51466
SWAG	0.824	0.735	0.759	0.885	75544
PCA + ESS (SI)	0.869	0.482	0.804	0.901	98696
MC dropout	0.872	0.554	0.775	0.914	25733
SGD	0.861	0.625	-	-	12866

Table 3. URSABench **large-scale** benchmark performance. For SGD, we use a pre-trained model available in PyTorch.

Inference	Accuracy ↑	NLL ↓	Robustness ↑	Uncertainty ↑	Training Time ↓
SGLD	0.768	0.927	0.832	0.851	23925
SGHMC	0.768	0.920	0.842	0.850	24243
cSGLD(6)	0.771	0.915	0.836	0.849	45675
cSGHMC(6)	0.771	0.910	0.838	0.849	46282
cSGLD(3)	0.769	0.920	0.830	0.848	28275
cSGHMC(3)	0.769	0.919	0.830	0.847	28650
SWAG	0.733	1.056	0.820	0.845	67022
PCA + ESS (SI)	0.765	0.948	0.813	0.847	110205
MC dropout	0.760	0.951	0.804	0.848	10825
SGD	0.760	0.962	-	-	-

Table 4. URSABench edge performance-latency comparison. Here, # models indicate the maximum number of models runnable on the Jetson TX2 device for SGHMC. Ensm. size indicates the size of the original ensemble.

Dataset-Model	Inference	Accuracy ↑	NLL ↓	Robustness ↑	Uncertainty ↑	Latency ↓	# Models/Ensm. size
MNIST - MLP200	SGHMC	0.985	0.047	0.947	0.980	0.080	100/100
	MC dropout	0.984	0.048	0.945	0.980	0.017	1
	SGD	0.986	0.045	0.948	0.980	0.0006	1
	Distilled SGHMC	0.986	0.044	0.960	0.970	0.0006	1
CIFAR10 - ResNet50	SGHMC	0.949	0.255	0.765	0.929	0.352	49/50
	MC dropout	0.948	0.208	0.817	0.947	0.051	1
	SGD	0.943	0.274	0.787	0.937	0.007	1
	Distilled SGHMC	0.932	0.279	0.787	0.929	0.007	1
CIFAR10 - WideResNet28x10	SGHMC	0.966	0.140	0.809	0.943	0.154	12/50
	MC dropout	0.957	0.158	0.805	0.947	0.031	1
	SGD	0.963	0.138	0.815	0.941	0.013	1
	Distilled SGHMC	0.950	0.169	0.812	0.946	0.013	1
CIFAR100 - ResNet50	SGHMC	0.738	1.222	0.780	0.872	0.348	49/50
	MC dropout	0.786	1.006	0.776	0.878	0.071	1
	SGD	0.732	1.302	0.764	0.872	0.007	1
	Distilled SGHMC	0.732	1.088	0.794	0.871	0.007	1
CIFAR100 - WideResNet28x10	SGHMC	0.811	0.784	0.783	0.884	0.153	12/50
	MC dropout	0.798	0.846	0.730	0.887	0.053	1
	SGD	0.806	0.785	0.776	0.874	0.013	1
	Distilled SGHMC	0.785	0.839	0.812	0.877	0.013	1
ImageNet - ResNet50	SGHMC	0.768	0.920	0.844	0.862	0.085	6/6
	MC dropout	0.760	0.951	0.804	0.848	0.076	1
	SGD	0.760	0.962	0.835	0.861	0.014	1
	Distilled SGHMC	0.736	1.086	0.865	0.849	0.014	1

the compute time, which shows HMC to be significantly less time-consuming. This is due to the ability to fit all the data and model parameters on the GPU. Recall that the training-time scalability represents the total no. of seconds required during training. To benchmark this, we use an Nvidia TITAN X GPU, with a batch size of 128, and 4 PyTorch dataloader workers.

5.2 Medium-Scale Benchmark Results

The medium-scale results are displayed in Table 2. The detailed experimental results behind each composite score can again be found in Appendix E. We note that we provide two sets of results for cSGHMC and cSGLD, with different ensemble sizes, denoted by the number in parentheses. For getting 50 parameter samples from the posterior using either cSGLD or cSGHMC, we run many cycles, which results in very high training time as compared to SGLD and SGHMC. Thus, for a more fair comparison with SGLD and SGHMC, we limit the training time of cSGLD and cSGHMC to be similar to their non-cyclic versions by reducing the number of cycles and parameter samples. In the setting with the reduced number of cycles, we extract nine parameter samples, as compared to 50 for the full number of cycles. Overall, the medium-scale experiments indicate a slight improvement in predictive performance and decision-making tasks from both cSGHMC (50) and cSGLD (50) followed by PCA + ESS (SI) & cSGLD. Importantly, the cSGLD & cSGHMC results with a lower ensemble size do better overall compared to cSGLD and cSGHMC with 50 parameter samples. Thus, once again, a user may prefer using MC dropout, SGLD/SGHMC or their cyclic versions with fewer parameter samples extracted, as they provide respectable performance in significantly less time. This is due to the large proportion of time that the cyclic schemes spend exploring without sampling. Furthermore, if the goal is to compute uncertainty metrics and ultimately use them for misclassification detection or OOD detection, then SGHMC/SGLD provide better performance in most cases. Another important result that can be seen from the Tables 8, 11, 14 and 17 in Appendix E is the utility of the decision-making task in highlighting the top performing approximate inference schemes for each model and dataset, via its correlation with low NLL and high accuracy.

For benchmarking the training-time scalability, we use an Nvidia TITAN X GPU, with a batch size of 128, and 4 PyTorch dataloader workers. Note that we provide an estimated training time for the methods which involve stochastic gradients. More specifically, we compute the per-epoch average training time for methods involving stochastic gradients, and then extrapolate it by multiplying it with the total number of epochs involving the same stochastic gradient based computation in that method. For example, while computing training time for SGLD and cSGLD, we first compute the average training time for SGLD for the same model-dataset combination, and then extrapolate it to the average training time shown in this paper by multiplying it

by the total number of epochs run for cSGLD. We use the same approach for SGHMC and cSGHMC, as well as for SGD, MC Dropout, SWAG and PCA+SSI (ESS).

5.3 Large-Scale Benchmark Results

The results for the large-scale benchmark are presented in Table 3. The detailed experimental results behind each composite score can be found in Appendix E. Also, similar to the medium-scale benchmarks, we provide two sets of results for cSGLD and cSGHMC, with different number of parameter samples extracted, where the results corresponding to fewer parameter samples have very similar training time to their non-cyclic versions. The relative performance trend that we observe here is very similar to that of the medium-scale benchmark. In terms of performance across the board, cSGHMC and cSGLD outperform the rest of the methods. Furthermore, cSGLD and cSGHMC perform very similar to their non-cyclic counterparts, which can be potentially attributed to the less number of samples or the number of cycles run. However, we notice here that PCA + ESS (SI) and SWAG do not perform as well as they did in the previous set of experiments. Finally, due to the high training runtime requirement of the cSGHMC/cSGLD, a user might yet again prefer SGHMC or MC Dropout as they provide reasonable performance in significantly less time.

Given the larger size of the ImageNet dataset (more than 1.2 million training images), we use multi-GPU training with a larger batch size of 256, processed using 24 PyTorch dataloader workers. For benchmarking the training time for large-scale benchmark, we use 4 Nvidia Titan X GPUs. We provide an approximate training time for large-scale benchmark results, using the same technique described in the previous subsection.

5.4 Edge Run Time Results

In this experiment, we focus on SGHMC, its distilled version using the BDK algorithm, MC dropout, and the standard SGD point-estimated model. For the BDK algorithm, the student model architecture matches that of the teacher model. Recall that MC dropout requires us to store a single model, and the posterior predictive distribution is computed using multiple forward passes. Note that the 8GB physical memory of Jetson TX2 is shared by both the CPU and the GPU. As a result, we can only accommodate a limited number of samples from some models. We present the profiling results for the edge deployment are presented in Table 4.

As we can see, the number of model samples we can fit onto the Jetson TX2 varies by model type. Specifically, for WideResNet28x10, we can only fit 12 samples from the ensemble of 50 models. For ImageNet, the large model size means we were only able to deploy six samples onto the board. The results of SGHMC are computed using only the model samples that we could fit on the Jetson TX2. Additionally, for the point-estimated model and the distilled model, we excluded the model uncertainty from the uncertainty and robustness scores of the composite score, as model uncertainty requires an expectation over multiple samples of an ensemble.

For MNIST, we can clearly see that the performance of SGD or distilled SGHMC across all four composite scores is very similar to that of SGHMC and MC dropout, making them an attractive choice considering the latency values. For WideResNet28x10 on CIFAR10 dataset, we observe that SGD point-estimated models tend to perform better than its counterparts while looking at NLL, and have similar performance on other composite scores during comparison. It also has relatively very low latency. For ResNet50 on CIFAR datasets, the trade-off between different performance metrics and latency is high. MC dropout tends to perform best across all four composite scores against the other three approximate inference methods in this case, but it comes at a higher latency cost. Finally, for ImageNet, we observe that while SGHMC outperforms other methods across all the scores, the trade-off between latency and performance is not as significant when compared against SGD. In this case, a user might select the SGD point-estimated model for practical applications, with a small degradation in performance.

5.5 Discussion

We make several observations based on the benchmark results presented in this section. First, if we are primarily interested in the accuracy, robustness or uncertainty metrics, then cSGHMC and cSGLD (using more parameter samples) are often the best choice. Across the different benchmarks, we consistently see them outperforming other methods (see Table 1, 2, 3). Second, MC Dropout, SGHMC/SGLD and cSGHMC/cSGLD (using fewer parameter samples) can often provide reasonable uncertainty and accuracy performance for lower training time budgets. Third, PCA-based subspace inference and SWAG require more training time than SGLD and SGHMHC as can be seen in Tables 1, 2, and 3. This is largely due to the time needed to generate the subspace (for subspace inference) or the Gaussian approximation to the parameter posterior (for SWAG). Furthermore, PCA-based subspace inference and SWAG do not provide consistent performance improvement over the rest of the methods in terms of accuracy, NLL, robustness, and uncertainty based on these results.

For the large-scale benchmark, we also see that the SGD based point-estimated models are able to compete with the posterior ensemble approaches. This may be due to the limited number of MCMC samples used or greater relative difficulty in mixing and requires further study. For the edge deployment scenario considered, we see that MC

dropout is a good alternative to SGHMC in terms of the tradeoff between prediction performance and prediction latency. It is also important to note that the other approximate inference techniques discussed in this paper, apart from the ones in Table 4, have similar latency properties to SGHMC due to performing a forward pass for each of a set of parameter samples. Finally, while the distillation approach is interesting, we observed that it does not perform better than the SGD-based models in the medium and large-scale benchmark settings. Further study is needed to determine if optimizing student model size as in Vadera et al. (2020) can lead to improved tradeoffs.

6 Conclusion and Future Work

This paper describes URSABench, a system for benchmarking multiple aspects of the performance of approximate Bayesian inference methods for deep neural networks. We hope that the development of this benchmarking system and its components will facilitate research in the domain of approximate Bayesian inference by better exposing the performance trade-offs achieved by different methods in terms of uncertainty, robustness, scalability and accuracy. We believe the simultaneous assessment of these properties is critical to better understand which methods are most effective on different downstream tasks and in different deployment contexts (e.g., server and edge).

A further challenge in the development of this system is ensuring a fair comparison between approaches. This can be difficult for new model/method/data set combinations without established hyperparameters, requiring careful hyperparameter optimization. This highlights the more complex issue of how to fairly benchmark the end-to-end process of hyperparameter optimization and inference in terms of computational resource use. In this work, the training time scalability measurements are assessed on tuned models only. The cost of hyperparameter optimization is not reflected, and thus methods with more hyperparameters that may take longer to optimize for new problems and tasks are not penalized in these results. This issue requires further study.

Acknowledgements

Research reported in this paper was sponsored in part by the CCDC Army Research Laboratory under Cooperative Agreement W911NF-17-2-0196 (ARL IoBT CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

Balan, A. K., Rathod, V., Murphy, K. P., and Welling, M. Bayesian dark knowledge. In *NeurIPS*, 2015.

Balandat, M., Karrer, B., Jiang, D. R., Daulton, S., Letham, B., Wilson, A. G., and Bakshy, E. Botorch: Programmable bayesian optimization in pytorch. *arXiv preprint arXiv:1910.06403*, 2019.

Bottou, L. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pp. 177–186. Springer, 2010.

Brier, G. W. Verification of forecasts expressed in terms of probability. *Monthly weather review*, 78(1):1–3, 1950.

Carlini, N. and Wagner, D. Towards evaluating the robustness of neural networks. *2017 IEEE Symposium on Security and Privacy (SP)*, 2017.

Casella, G. and George, E. I. Explaining the gibbs sampler. *The American Statistician*, 46(3):167–174, 1992.

Chen, T., Fox, E. B., and Guestrin, C. Stochastic gradient hamiltonian monte carlo. In *ICML*, 2014.

Chib, S. and Greenberg, E. Understanding the metropolis-hastings algorithm. *The american statistician*, 49(4): 327–335, 1995.

Clanuwat, T., Bober-Irizar, M., Kitamoto, A., Lamb, A., Yamamoto, K., and Ha, D. Deep learning for classical japanese literature, 2018.

Coates, A., Ng, A. Y., and Lee, H. An analysis of single-layer networks in unsupervised feature learning. In *AISTATS*, 2011.

Cobb, A. D. *The practicalities of scaling Bayesian neural networks to real-world applications*. PhD thesis, University of Oxford, 2020.

Cobb, A. D., Roberts, S. J., and Gal, Y. Loss-calibrated approximate inference in Bayesian neural networks. *Theory of deep learning workshop, ICML*, 2018.

Cobb, A. D., Baydin, A. G., Markham, A., and Roberts, S. J. Introducing an Explicit Symplectic Integration Scheme for Riemannian Manifold Hamiltonian Monte Carlo. *arXiv preprint arXiv:1910.06243*, 2019.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Depeweg, S., Hernandez-Lobato, J.-M., Doshi-Velez, F., and Udluft, S. Decomposition of uncertainty in Bayesian deep learning for efficient and risk-sensitive learning. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1184–1193. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/depeweg18a.html.

Duane, S., Kennedy, A. D., Pendleton, B. J., and Roweth, D. Hybrid monte carlo. *Physics letters B*, 195(2):216–222, 1987.

Dusenberry, M., Jerfel, G., Wen, Y., Ma, Y., Snoek, J., Heller, K., Lakshminarayanan, B., and Tran, D. Efficient and scalable Bayesian neural nets with rank-1 factors. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 2782–2792. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/dusenberry20a.html.

Gal, Y. and Ghahramani, Z. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *ICML*, 2016.

Ghosh, S., Delle Fave, F. M., and Yedidia, J. Assumed density filtering methods for learning bayesian neural networks. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

Girolami, M. and Calderhead, B. Riemann manifold langevin and hamiltonian monte carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(2):123–214, 2011.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. On calibration of modern neural networks. In *ICML*, pp. 1321–1330, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

Izmailov, P., Maddox, W. J., Kirichenko, P., Garipov, T., Vetrov, D. P., and Wilson, A. G. Subspace inference for bayesian deep learning. In *UAI*, 2019.

Jaakkola, T. S. and Jordan, M. I. Bayesian parameter estimation via variational methods. *Statistics and Computing*, 10(1):25–37, 2000.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. Technical report, 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

LeCun, Y. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Li, Y. and Turner, R. E. Rényi divergence variational inference. In *Advances in Neural Information Processing Systems 29*, pp. 1073–1081. 2016.

Louizos, C. and Welling, M. Multiplicative normalizing flows for variational bayesian neural networks. In *ICML*, 2017.

MacKay, D. J. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

Maddox, W., Garipov, T., Izmailov, P., Vetrov, D. P., and Wilson, A. G. A simple baseline for bayesian uncertainty in deep learning. In *NeurIPS*, 2019.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018.

Malinin, A., Mlodozeniec, B., and Gales, M. Ensemble distribution distillation. In *ICLR*, 2020.

Minka, T. P. Expectation propagation for approximate bayesian inference. In *UAI*, 2001.

Murray, I., Adams, R. P., and MacKay, D. J. C. Elliptical slice sampling. In *AISTATS*, 2010.

Neal, R. M. *Bayesian Learning for Neural Networks*. Springer-Verlag, 1996.

Neal, R. M. Slice sampling. *Annals of statistics*, pp. 705–741, 2003.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading digits in natural images with unsupervised feature learning. 2011.

Niculescu-Mizil, A. and Caruana, R. Predicting good probabilities with supervised learning. In *ICML*, pp. 625–632, 2005.

NVIDIA Corporation. TensorRT Developer Guide. URL https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html.

Ovadia, Y., Fertig, E., Ren, J., Nado, Z., Sculley, D., Nowozin, Sebastian Dillon, J., Lakshminarayanan, B., and Snoek, J. Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift. In *NeurIPS*, pp. 13969–13980, 2019.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Smith, A. F. and Roberts, G. O. Bayesian computation via the gibbs sampler and related markov chain monte carlo methods. *Journal of the Royal Statistical Society: Series B (Methodological)*, 55(1):3–23, 1993.

Vadera, M. P., Jalaian, B., and Marlin, B. M. Generalized bayesian posterior expectation distillation for deep neural networks. In *UAI*, 2020.

Wang, H., Ge, S., Lipton, Z., and Xing, E. P. Learning robust global representations by penalizing local predictive power. In *Advances in Neural Information Processing Systems*, pp. 10506–10518, 2019.

Wang, K.-C., Vicol, P., Lucas, J., Gu, L., Grosse, R., and Zemel, R. Adversarial distillation of Bayesian neural network posteriors. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5190–5199. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/wang18i.html.

Welling, M. and Teh, Y. W. Bayesian learning via stochastic gradient langevin dynamics. In *ICML*, pp. 681–688, 2011.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *ArXiv*, abs/1708.07747, 2017.

Zagoruyko, S. and Komodakis, N. Wide residual networks. In Richard C. Wilson, E. R. H. and Smith, W. A. P. (eds.), *Proceedings of the British Machine Vision Conference (BMVC)*, pp. 87.1–87.12. BMVA Press, September 2016. ISBN 1-901725-59-6. doi: 10.5244/C.30.87. URL https://dx.doi.org/10.5244/C.30.87.

Zhang, R., Li, C., Zhang, J., Chen, C., and Wilson, A. G. Cyclical stochastic gradient mcmc for bayesian deep learning. In *ICLR*, 2020.

A DECISION-MAKING TASK

Bayesian decision theory takes Monte Carlo samples and averages them over a predetermined cost function, $\mathcal{C}(h, y)$ to result in an expected risk:

$$R(h|\mathbf{x}) \approx \frac{1}{S}\sum_{s=1}^{S} p(y|\mathbf{x}, \theta_s)\mathcal{C}(h, y).$$

The expected risk is a function of the decision, h, whereby the Bayesian optimal decision, h^* minimises the risk:

$$h^* = \underset{h}{\operatorname{argmin}}\, R(h|\mathbf{x}).$$

Once we have applied Bayesian decision theory to find an h^* for every input $\mathbf{x}$, for supervised classification, we can then determine the true cost of the decision taken by averaging over the test data (i.e. $\frac{1}{N}\sum_{n=1}^{N}\mathcal{C}(y_n^{\text{True}}, h_n^*)$), where the arguments have been reversed such that we compare the true label y_n^{True}, with the Bayesian optimal decision h_n^* (i.e. what cost did we actually have to pay when we took decision h_n^* for labelling y_n, when it was in fact y_n^{True}). A more detailed discussion can be seen in Ch. 4 of Cobb (2020).

The purpose of the decision-making task is to penalise inference schemes that provide poor calibrated uncertainty over the rarer (and hence more uncertain) classes. In particular, for MNIST, we retrain our models over a highly imbalanced data set, where 99% of the labels corresponding to classes 3 and 7 are removed. However, we then use the predictive distribution to with a predefined cost function to select the Bayes optimal decision for each predicted label. We then calculate the expected decision cost by averaging over the costs attributed to each decision compared to the true label. False negatives of the less frequent classes are penalised 1000 times more than false positives for the rest of the classes.

The small-scale setting for the decision-making task requires retraining over an imbalanced training set. However, for the medium-scale task we limit ourselves to using the same materialised samples from the balanced data set (although we expect to extend this to imbalanced training data in future work). We define our cost matrix to penalise false negatives 10 times as much as false positives. In particular, the task for the CIFAR10 penalises planes, automobiles, ships and trucks with a cost of 1.0 for false negatives and 0.1 for false positives. All other errors are penalised with 0.1 and correct decisions accrue zero cost. The same cost structure applies to CIFAR100, where tanks, rockets and pick-up trucks are

deemed the critical classes. For ImageNet, the following class indices as penalised with a cost of 100.0 for false negatives, and 0.1 for false positives: [403, 479, 565, 705, 751, 817, 847 864, 895] [3]. All other errors are penalised with 0.1 and correct decisions accrue zero cost.

The decision costs in Tables 8, 11, 14 and 17 demonstrate the utility of this task as they show a correlation with the NLL and the accuracy across all models and data sets.

B Uncertainty Decomposition for downstream tasks

The posterior predictive distribution is not the only statistic of the posterior distribution that is of interest. The decomposition of posterior uncertainty has also received recent attention in the literature. For example, Depeweg et al. (2018) and Malinin et al. (2020) describe the decomposition of the entropy of the posterior predictive distribution (the *total uncertainty*) into *expected data uncertainty* and *knowledge uncertainty*. These three forms of uncertainty are related by the equation shown below:

$$\underbrace{\mathcal{I}\left[y,\theta|\boldsymbol{x},\mathcal{D}\right]}_{\text{Knowledge Uncertainty}} = \underbrace{\mathcal{H}\left[\mathbb{E}_{p(\theta|\mathcal{D})}\left[p\left(y|\boldsymbol{x},\theta\right)\right]\right]}_{\text{Total Uncertainty}} - \underbrace{\mathbb{E}_{p(\theta|\mathcal{D})}\left[\mathcal{H}\left[p\left(y|\boldsymbol{x},\theta\right)\right]\right]}_{\text{Expected Data Uncertainty}} \quad (4)$$

Total uncertainty, as the name suggests, measures the total uncertainty in a prediction. Expected data uncertainty measures the uncertainty arising from class overlap. Knowledge uncertainty corresponds to the conditional mutual information between labels and model parameters and measures the disagreement between different models in the posterior. However, it can be efficiently computed as the difference between total uncertainty and expected data uncertainty, both of which are (functions) of posterior expectations. In recent work, Wang et al. (2018), Malinin et al. (2020) and Vadera et al. (2020) have leveraged this decomposition to explore a range of down-stream tasks that rely on uncertainty quantification and decomposition.

C Composite Score Breakdown

As alluded to in the main text, we build composite scores for robustness and uncertainty. The robustness relies on averaging both the total uncertainty AUROC and the model uncertainty AUROC over the OOD data sets. We then average once again over the mean total uncertainty and model uncertainty. The uncertainty composite score is built from the average misclassification AUROCs (e.g. the first three columns of Tables 7, 10, 13, 16, 19). For the medium-scale experiment, the uncertainty score is then averaged across CIFAR10 and CIFAR100 as well as ResNet50 and WideResNet28x10.

[3]The mapping of class indices referenced is available at: https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a

D Implementation Details

In this section, we describe the implementation details for the different inference methods used in our benchmark. It must be noted that for all inference methods using ResNet50 and WideResNet28x10 models, we use a pretrained SGD solution to warm-start our samplers. This is a standard pretraining procedure followed to make the methods more competitive (Maddox et al., 2019). Further, the ensemble size is set to 100 for MNIST dataset, 50 for CIFAR datasets, and 6 for ImageNet dataset. When applicable, we always collect a sample at the end of an epoch. The difference in ensemble size is due to the large amounts of computational requirements for training ResNet50 and WideResNet28x10 on CIFAR datasets, as well training ResNet50 on ImageNet. While tuning hyperparameters for MNIST, we apply Bayesian optimization with a limit of 200 evaluations for each approach (Balandat et al., 2019). On the other hand, for CIFAR datasets, we refer to existing literature and use the same hyperparameters if directly applicable, or search around the hyperparameters obtained for similar models and datasets.

SGLD: For CIFAR datasets, we use a burn-in of 100 epochs and initial learning rates of 0.1 for WideResNet28x10 model and 0.05 for ResNet50. The prior std. dev. is set to 0.5 for both the cases. We decay the learning rate using cosine annealing schedule to its half value by the end of sampling. For ImageNet datasets, we use a burn-in of 5 epochs, initial learning rates of 0.05, and set the prior std. dev. to 0.1. For MNIST, the optimal hyperparameter values obtained are: initial learning rate of 0.099, prior std. dev. of 0.16 and 50 burn in epochs.

SGHMC: We use the same hyperparameters and learning rate schedule as described for SGLD for the CIFAR and ImageNet datasets. Additionally, we set the friction term to 0.5 (Chen et al., 2014). This is equivalent to the α term shown in Zhang et al. (2020). For MNIST, the optimal hyperparameter values obtained are: initial learning rate of 0.03, prior std. dev. of 0.14, 50 burn in epochs, and friction term set to 0.1.

cSGHMC: We use the same hyperparameters given in Zhang et al. (2020) for CIFAR datasets. For ImageNet, we use the initial learning rate for a cycle to 0.01, prior std. dev. of 0.1, cycle length of 7 epochs, of which 4 epochs are used for SGD-exploration phase, 1 epoch for burn-in,

and collect 2 samples from each of the last two epochs of the cycle. For ImageNet, we run a total of 3 cycles. For MNIST, the optimal hyperparameter values obtained are: initial learning rate of 0.06, prior std. dev. of 0.33, cycle length of 22 epochs, of which 17 epochs are used for SGD-exploration phase, and samples are collected from the last 4 epochs, and friction term set to 0.21.

cSGLD: We use the same hyperparameters given in Zhang et al. (2020) for CIFAR datasets. For ImageNet, we use the same hyperparameters described for cSGHMC. For MNIST, the optimal hyperparameter values obtained are: initial learning rate of 0.06, prior std. dev. of 0.33, cycle length of 22 epochs, of which 17 epochs are used for SGD-exploration phase, and samples are collected from the last 4 epochs, and friction term set to 0.21.

SWAG: We use the same hyperparameters given in Izmailov et al. (2019) for CIFAR models, except that we set the weight decay for ResNet models to 10^{-4} and borrow its remaining hyperparameters from WideResNet28x10. This means that we utilize last 20 SGD iterates to find parameters for the Gaussian approximation to the mode. For ImageNet, we set the exploration learning rate to 0.005, and start collecting samples for constructing the Gaussian approximation after 10 epochs. We collect 20 parameter samples for generating the Gaussian approximation. Furthermore, the variant of SWAG used in our benchmark is SWAG-diagonal.

PCA + ESS (SI): We use the same hyperparameters given in Izmailov et al. (2019) for CIFAR models, except that we set the weight decay for ResNet models to 10^{-4} and borrow its remaining hyperparameters from WideResNet28x10. We construct a subspace of rank 20 for all models and datasets. For ImageNet, we set the exploration learning rate to 0.005, and start collecting samples for constructing the subspace after 10 epochs. We collect 20 parameter samples for generating the subspace of rank 20. For MNIST, we start with an initial learning rate of 0.04 and decay it 0.002 over 50 epochs. Further, we run SGD at the same learning rate for another 50 epochs and collect the iterates from each of the final 20 epochs to construct our PCA subspace. The momentum for our SGD optimizer is set to 0.54 through the entire run. For all the dataset and model combinations, we use elliptical slice sampling (Murray et al., 2010) on the low rank PCA subspace with a prior of 2. and a temperature of 5000.

MC Dropout: For all the models on CIFAR & MNIST datasets, we use a dropout of 0.2 before the final linear layer. For ImageNet, we use a dropout rate of 0.05 just before the final layer. We always fine-tune after warm-starting with SGD point-estimated model.

E Additional Experimental Results

Additional experimental results are provided in Tables 5 - 22.

Table 5. Comparison of predictive performance and decision making cost while using an MLP $[784, 200, 200, 10]$ on MNIST. Results presented as mean $\pm$ std. dev. across 5 trials.

Inference	Accuracy ↑	NLL ↓	BS ↓	ECE ↓	Decision Cost ↓	Training Time ↓
HMC	$98.19 \pm 0.10\%$	0.0593 ± 0.0016	0.0280 ± 0.0008	0.0079 ± 0.0008	7101 ± 346	178 ± 1.5
SGLD	$98.39 \pm 0.04\%$	0.0492 ± 0.0022	0.0236 ± 0.0005	0.0041 ± 0.0024	5410 ± 778	765 ± 4.2
SGHMC	$98.62 \pm 0.03\%$	0.0446 ± 0.0003	0.0210 ± 0.0002	0.0073 ± 0.0004	5408 ± 240	768 ± 2.3
cSGLD	$98.57 \pm 0.03\%$	0.0476 ± 0.0011	0.0223 ± 0.0003	0.0056 ± 0.0003	6526 ± 2241	1063 ± 4.2
cSGHMC	$98.36 \pm 0.09\%$	0.0533 ± 0.0016	0.0256 ± 0.0010	0.0033 ± 0.0003	4824 ± 1855	1111 ± 3.3
PCA + ESS (SI)	$98.40 \pm 0.07\%$	0.0520 ± 0.0016	0.0251 ± 0.0007	0.0036 ± 0.0005	3809 ± 1150	1146 ± 10.2
MC dropout	$98.58 \pm 0.07\%$	0.0501 ± 0.0031	0.0218 ± 0.0008	0.0042 ± 0.0006	15236 ± 1184	377 ± 2.4
SGD	$98.60 \pm 0.02\%$	0.0452 ± 0.0012	0.0213 ± 0.0003	0.0032 ± 0.0005	8972 ± 720	193 ± 0.9

Table 6. Comparison of OOD detection performance while using an MLP $[784, 200, 200, 10]$ on MNIST. Results presented as mean $\pm$ std. dev. across 5 trials.

Inference	OOD Dataset	AUROC- Model Uncertainty ↑	AUROC - Total Uncertainty ↑
HMC	Fashion MNIST	0.966 ± 0.013	0.946 ± 0.017
	KMNIST	0.968 ± 0.013	0.948 ± 0.017
SGLD	Fashion MNIST	0.867 ± 0.110	0.944 ± 0.005
	KMNIST	0.871 ± 0.103	0.944 ± 0.005
SGHMC	Fashion MNIST	0.933 ± 0.009	0.953 ± 0.010
	KMNIST	0.932 ± 0.009	0.952 ± 0.010
cSGLD	Fashion MNIST	0.954 ± 0.004	0.950 ± 0.005
	KMNIST	0.954 ± 0.004	0.950 ± 0.005
cSGHMC	Fashion MNIST	0.923 ± 0.021	0.931 ± 0.017
	KMNIST	0.923 ± 0.020	0.933 ± 0.017
PCA + ESS (SI)	Fashion MNIST	0.933 ± 0.006	0.938 ± 0.009
	KMNIST	0.934 ± 0.006	0.940 ± 0.009
MC dropout	Fashion MNIST	0.942 ± 0.013	0.943 ± 0.010
	KMNIST	0.943 ± 0.013	0.944 ± 0.010
SGD	Fashion MNIST	-	0.945 ± 0.010
	KMNIST	-	0.943 ± 0.010

Table 7. Comparison of misclassification detection while using an MLP $[784, 200, 200, 10]$ on MNIST.

Inference	AUROC- Model Uncertainty ↑	AUROC - Total Uncertainty ↑	AUROC- Model Confidence ↑	AUCPR- Model Uncertainty ↑	AUCPR - Total Uncertainty ↑	AUCPR- Model Confidence ↑
HMC	0.9706	0.9734	0.9743	0.3429	0.3888	0.4145
SGLD	0.9739	0.9800	0.9800	0.3530	0.4502	0.4632
SGHMC	0.9786	0.9815	0.9823	0.3546	0.3929	0.4131
cSGLD	0.9769	0.9801	0.9798	0.3695	0.4477	0.4478
cSGHMC	0.9730	0.9786	0.9786	0.3260	0.4255	0.4404
PCA + ESS (SI)	0.9539	0.9774	0.9772	0.2059	0.4298	0.4248
MC dropout	0.9754	0.9763	0.976	0.4085	0.4300	0.4199
SGD	-	0.9795	0.9794	-	0.4273	0.4389

Table 8. Comparison of predictive performance and decision making cost while using ResNet50 on CIFAR10.

Inference	Accuracy ↑	NLL ↓	BS ↓	ECE ↓	Decision Cost ↓	Training Time ↓
SGLD	0.943	0.275	0.094	0.041	176.000	31193
SGHMC	0.949	0.256	0.086	0.035	154.000	31538
cSGLD(50)	0.961	0.137	0.060	0.011	118.500	139336
cSGHMC(50)	0.962	0.124	0.056	0.009	125.000	141291
cSGLD(9)	0.952	0.198	0.075	0.023	147.200	31193
cSGHMC(9)	0.956	0.178	0.070	0.019	135.100	31538
SWAG	0.931	0.311	0.114	0.047	200.900	44344
PCA + ESS (SI)	0.949	0.174	0.080	0.027	166.600	55564
MC dropout	0.948	0.208	0.083	0.032	159.500	15768
SGD	0.943	0.274	0.095	0.040	171.700	7884

Table 9. Comparison of OOD detection performance while using ResNet50 on CIFAR10.

Inference	OOD Dataset	AUROC- Model Uncertainty ↑	AUROC - Total Uncertainty ↑
SGLD	STL10	0.630	0.679
	SVHN	0.863	0.876
SGHMC	STL10	0.639	0.680
	SVHN	0.855	0.849
cSGLD(50)	STL10	0.678	0.700
	SVHN	0.952	0.953
cSGHMC(50)	STL10	0.668	0.692
	SVHN	0.960	0.960
cSGLD(9)	STL10	0.666	0.687
	SVHN	0.902	0.908
cSGHMC(9)	STL10	0.667	0.689
	SVHN	0.936	0.939
SWAG	STL10	0.618	0.671
	SVHN	0.878	0.908
PCA + ESS (SI)	STL10	0.673	0.677
	SVHN	0.949	0.947
MC dropout	STL10	0.665	0.695
	SVHN	0.926	0.938
SGD	STL10	N/A	0.682
	SVHN	N/A	0.892

Table 10. Comparison of Misclassification detection while using ResNet50 on CIFAR10.

Inference	AUROC- Model Uncertainty ↑	AUROC - Total Uncertainty ↑	AUROC- Model Confidence ↑	AUCPR- Model Uncertainty ↑	AUCPR - Total Uncertainty ↑	AUCPR- Model Confidence ↑
SGLD	0.936	0.934	0.935	0.466	0.483	0.487
SGHMC	0.930	0.929	0.929	0.406	0.411	0.418
cSGLD(50)	0.949	0.950	0.951	0.398	0.403	0.435
cSGHMC(50)	0.948	0.952	0.954	0.394	0.422	0.459
cSGLD(9)	0.939	0.943	0.943	0.378	0.435	0.448
cSGHMC(9)	0.935	0.940	0.940	0.353	0.405	0.420
SWAG	0.890	0.927	0.927	0.418	0.479	0.472
PCA + ESS (SI)	0.932	0.934	0.941	0.391	0.419	0.472
MC dropout	0.946	0.947	0.947	0.455	0.485	0.477
SGD	0.523	0.937	0.936	0.151	0.464	0.456

Table 11. Comparison of predictive performance and decision making cost while using ResNet50 on CIFAR100.

Inference	Accuracy ↑	NLL ↓	BS ↓	ECE ↓	Decision Cost ↓	Training Time ↓
SGLD	0.736	1.254	0.400	0.141	299.800	30153
SGHMC	0.738	1.222	0.397	0.137	295.000	30468
cSGLD(50)	0.770	0.902	0.324	0.066	250.100	136891
cSGHMC(50)	0.782	0.838	0.306	0.053	241.100	137106
cSGLD(9)	0.779	0.858	0.316	0.047	242.900	30153
cSGHMC(9)	0.794	0.794	0.295	0.048	223.700	30468
SWAG	0.735	1.221	0.400	0.135	295.200	44343
PCA + ESS (SI)	0.761	0.920	0.335	0.032	261.200	60880
MC dropout	0.786	1.006	0.330	0.115	233.100	15234
SGD	0.732	1.302	0.408	0.148	303.700	7617

Table 12. Comparison of OOD detection performance while using ResNet50 on CIFAR100.

Inference	OOD Dataset	AUROC- Model Uncertainty ↑	AUROC - Total Uncertainty ↑
SGLD	STL10	0.750	0.772
	SVHN	0.746	0.784
SGHMC	STL10	0.750	0.777
	SVHN	0.754	0.784
cSGLD(50)	STL10	0.786	0.800
	SVHN	0.820	0.834
cSGHMC(50)	STL10	0.793	0.811
	SVHN	0.808	0.827
cSGLD(9)	STL10	0.767	0.804
	SVHN	0.813	0.846
cSGHMC(9)	STL10	0.783	0.814
	SVHN	0.809	0.825
SWAG	STL10	0.748	0.778
	SVHN	0.732	0.771
PCA + ESS (SI)	STL10	0.779	0.797
	SVHN	0.816	0.807
MC dropout	STL10	0.785	0.801
	SVHN	0.755	0.752
SGD	STL10	0.500	0.765
	SVHN	0.500	0.763

Table 13. Comparison of Misclassification detection while using ResNet50 on CIFAR100.

Inference	AUROC- Model Uncertainty ↑	AUROC - Total Uncertainty ↑	AUROC- Model Confidence ↑	AUCPR- Model Uncertainty ↑	AUCPR - Total Uncertainty ↑	AUCPR- Model Confidence ↑
SGLD	0.866	0.873	0.872	0.635	0.677	0.675
SGHMC	0.864	0.872	0.871	0.622	0.668	0.666
cSGLD(50)	0.875	0.879	0.883	0.633	0.650	0.661
cSGHMC(50)	0.877	0.881	0.885	0.617	0.639	0.651
cSGLD(9)	0.851	0.871	0.876	0.530	0.617	0.636
cSGHMC(9)	0.860	0.876	0.882	0.558	0.612	0.634
SWAG	0.855	0.870	0.869	0.625	0.671	0.667
PCA + ESS (SI)	0.858	0.863	0.877	0.618	0.634	0.667
MC dropout	0.875	0.880	0.877	0.613	0.639	0.624
SGD	N/A	0.873	0.870	N/A	0.687	0.680

Table 14. Comparison of predictive performance and decision making cost while using WideResNet28x10 on CIFAR10.

Inference	Accuracy ↑	NLL ↓	BS ↓	ECE ↓	Decision Cost ↓	Training Time ↓
SGLD	0.963	0.153	0.060	0.023	121.300	72365
SGHMC	0.967	0.140	0.055	0.019	106.500	72740
cSGLD(50)	0.967	0.105	0.049	0.009	107.300	325205
cSGHMC(50)	0.962	0.122	0.057	0.008	125.100	327330
cSGLD(9)	0.961	0.128	0.060	0.008	132.500	72365
cSGHMC(9)	0.951	0.182	0.076	0.019	150.000	72740
SWAG	0.919	0.260	0.121	0.028	270.000	106747
PCA + ESS (SI)	0.951	0.177	0.082	0.054	163.100	141980
MC dropout	0.957	0.158	0.067	0.019	149.000	36370
SGD	0.963	0.138	0.060	0.018	117.100	18185

Table 15. Comparison of OOD detection performance while using WideResNet28x10 on CIFAR10.

Inference	OOD Dataset	AUROC- Model Uncertainty ↑	AUROC - Total Uncertainty ↑
SGLD	STL10	0.665	0.660
	SVHN	0.936	0.951
SGHMC	STL10	0.665	0.662
	SVHN	0.943	0.954
cSGLD(50)	STL10	0.678	0.679
	SVHN	0.966	0.968
cSGHMC(50)	STL10	0.679	0.690
	SVHN	0.963	0.960
cSGLD(9)	STL10	0.672	0.674
	SVHN	0.944	0.958
cSGHMC(9)	STL10	0.667	0.682
	SVHN	0.924	0.925
SWAG	STL10	0.649	0.667
	SVHN	0.914	0.943
PCA + ESS (SI)	STL10	0.663	0.673
	SVHN	0.897	0.970
MC dropout	STL10	0.672	0.688
	SVHN	0.897	0.922
SGD	STL10	N/A	0.667
	SVHN	N/A	0.963

Table 16. Comparison of Misclassification detection while using WideResNet28x10 on CIFAR10.

Inference	AUROC- Model Uncertainty ↑	AUROC - Total Uncertainty ↑	AUROC- Model Confidence ↑	AUCPR- Model Uncertainty ↑	AUCPR - Total Uncertainty ↑	AUCPR- Model Confidence ↑
SGLD	0.946	0.946	0.946	0.416	0.445	0.438
SGHMC	0.941	0.940	0.940	0.370	0.401	0.398
cSGLD(50)	0.955	0.957	0.958	0.406	0.416	0.438
cSGHMC(50)	0.946	0.950	0.951	0.393	0.436	0.476
cSGLD(9)	0.941	0.948	0.949	0.343	0.415	0.430
cSGHMC(9)	0.932	0.939	0.939	0.348	0.430	0.455
SWAG	0.900	0.915	0.916	0.375	0.468	0.468
PCA + ESS (SI)	0.918	0.931	0.948	0.337	0.387	0.478
MC dropout	0.946	0.947	0.947	0.432	0.467	0.467
SGD	N/A	0.941	0.942	N/A	0.390	0.387

Table 17. Comparison of predictive performance and decision making cost while using WideResNet28x10 on CIFAR100.

Inference	Accuracy ↑	NLL ↓	BS ↓	ECE ↓	Decision Cost ↓	Training Time ↓
SGLD	0.809	0.796	0.280	0.064	202.300	70760
SGHMC	0.810	0.779	0.277	0.061	203.100	71120
cSGLD(50)	0.827	0.666	0.248	0.030	183.100	319190
cSGHMC(50)	0.813	0.699	0.264	0.024	198.400	320040
cSGLD(9)	0.813	0.754	0.268	0.062	200.200	70760
cSGHMC(9)	0.790	0.868	0.304	0.072	222.700	71120
SWAG	0.710	1.149	0.414	0.110	316.900	106745
PCA + ESS (SI)	0.817	0.679	0.263	0.038	196.600	136360
MC dropout	0.798	0.846	0.293	0.081	214.300	35560
SGD	0.806	0.785	0.280	0.046	205.100	17780

Table 18. Comparison of OOD detection performance while using WideResNet28x10 on CIFAR100.

Inference	OOD Dataset	AUROC- Model Uncertainty ↑	AUROC - Total Uncertainty ↑
SGLD	STL10	0.791	0.819
	SVHN	0.790	0.786
SGHMC	STL10	0.788	0.824
	SVHN	0.760	0.741
cSGLD(50)	STL10	0.818	0.839
	SVHN	0.818	0.812
cSGHMC(50)	STL10	0.812	0.831
	SVHN	0.773	0.772
cSGLD(9)	STL10	0.804	0.829
	SVHN	0.784	0.782
cSGHMC(9)	STL10	0.792	0.817
	SVHN	0.772	0.792
SWAG	STL10	0.732	0.753
	SVHN	0.672	0.704
PCA + ESS (SI)	STL10	0.813	0.827
	SVHN	0.760	0.814
MC dropout	STL10	0.798	0.815
	SVHN	0.642	0.645
SGD	STL10	N/A	0.820
	SVHN	N/A	0.732

Table 19. Comparison of Misclassification detection while using WideResNet28x10 on CIFAR100.

Inference	AUROC- Model Uncertainty ↑	AUROC - Total Uncertainty ↑	AUROC- Model Confidence ↑	AUCPR- Model Uncertainty ↑	AUCPR - Total Uncertainty ↑	AUCPR- Model Confidence ↑
SGLD	0.875	0.885	0.889	0.543	0.620	0.634
SGHMC	0.877	0.882	0.888	0.544	0.598	0.621
cSGLD(50)	0.887	0.888	0.895	0.567	0.591	0.613
cSGHMC(50)	0.882	0.882	0.890	0.589	0.596	0.625
cSGHMC(9)	0.864	0.877	0.880	0.575	0.629	0.646
cSGLD(9)	0.880	0.892	0.897	0.564	0.630	0.647
SWAG	0.837	0.857	0.860	0.601	0.675	0.686
PCA + ESS (SI)	0.853	0.868	0.888	0.520	0.559	0.603
MC dropout	0.883	0.887	0.887	0.617	0.638	0.637
SGD	N/A	0.869	0.879	N/A	0.586	0.622

Table 20. Comparison of predictive performance while using ResNet50 on ImageNet.

Inference	Accuracy ↑	NLL ↓	BS ↓	ECE ↓	Decision Cost ↓	Training Time
SGLD	0.768	0.927	0.326	0.034	6889	23925
SGHMC	0.768	0.920	0.325	0.028	6898	24243
cSGLD(6)	0.771	0.915	0.322	0.030	6773	45675
cSGHMC(6)	0.771	0.910	0.322	0.029	7078	46282
cSGLD(3)	0.769	0.920	0.324	0.032	7181	28275
cSGHMC(3)	0.769	0.919	0.324	0.031	7087	28650
SWAG	0.733	1.056	0.367	0.038	8089	67022
PCA + ESS (SI)	0.765	0.948	0.356	0.050	6782	110205
MCdropout	0.760	0.951	0.335	0.026	7152	10825
SGD	0.760	0.962	0.336	0.038	7431	-

Table 21. Comparison of OOD detection performance while using ResNet50 on ImageNet.

Inference	OOD Dataset	AUROC- Model Uncertainty ↑	AUROC - Total Uncertainty ↑
SGLD	ImageNet-Sketch	0.828	0.837
SGHMC	ImageNet-Sketch	0.841	0.844
cSGLD(6)	ImageNet-Sketch	0.832	0.841
cSGHMC(6)	ImageNet-Sketch	0.833	0.843
cSGLD(3)	ImageNet-Sketch	0.822	0.837
cSGHMC(3)	ImageNet-Sketch	0.821	0.839
SWAG	ImageNet-Sketch	0.809	0.831
PCA + ESS (SI)	ImageNet-Sketch	0.800	0.827
MC dropout	ImageNet-Sketch	0.772	0.837
SGD	ImageNet-Sketch	N/A	0.835

Table 22. Comparison of Misclassification detection while using ResNet50 on ImageNet.

Inference	AUROC- Model Uncertainty ↑	AUROC - Total Uncertainty ↑	AUROC- Model Confidence ↑	AUCPR- Model Uncertainty ↑	AUCPR - Total Uncertainty ↑	AUCPR- Model Confidence ↑
SGLD	0.829	0.857	0.867	0.547	0.616	0.649
SGHMC	0.826	0.856	0.867	0.531	0.616	0.649
cSGLD(6)	0.825	0.856	0.866	0.529	0.612	0.644
cSGHMC(6)	0.825	0.856	0.867	0.530	0.610	0.644
cSGLD(3)	0.818	0.858	0.867	0.521	0.615	0.647
cSGHMC(3)	0.817	0.857	0.867	0.518	0.612	0.645
SWAG	0.821	0.851	0.863	0.565	0.644	0.677
PCA + ESS (SI)	0.816	0.858	0.866	0.550	0.649	0.673
MC dropout	0.826	0.853	0.864	0.533	0.617	0.650
SGD	N/A	0.856	0.865	N/A	0.622	0.653