DISCRETE BAYESIAN NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Bayesian Neural Networks (BNNs) are used for uncertainty estimation by considering a usually continuous probability distribution over the parameters of a neural network. In this paper, we show how continuous BNNs can be discretized and ranked on a validation dataset for achieving higher quality of uncertainty while also drastically reducing the number of computations. We propose a ranking process for the case where the true predictive distribution of the validation dataset is known as well as for the case where only the samples from the true distribution are given, as is common in real-life datasets. We evaluate the proposed Discrete Bayesian Neural Networks (DiBNNs) versions of Variational Neural Networks, Hypermodels, Deep Ensembles, Bayes By Backpropagation, Monte Carlo Dropout and Layer Ensembles on The Neural Testbed which provides true uncertainty for the test set. When the uncertainty targets are given for the validation dataset, DiBNNs achieve up to 2.1 times lower KL-divergence values and require an order of magnitude fewer computations than BNNs. If the validation dataset does not have uncertainty targets, DiBNNs achieve up to 2.6 times better KL-divergence values at lower samples, which is beneficial under a limited computational budget.

1 INTRODUCTION

Estimation of uncertainty in neural networks is a problem of expressing the lack of model's ability to provide a correct answer due to either epistemic (lack of knowledge) or aleatoric (lack of descriptive data) reasons. Deep learning methods attempt to express uncertainty in classification tasks by providing a distribution over the target classes, but they fail to provide a good quality of uncertainty due to the tendency to be overconfident in their results (Guo et al., 2017). In most other problems, uncertainty estimation is omitted, and research is focused on the accuracy of the model without the ability to answer *I do not know*. This leads to issues such as hallucinations in Large Language Models (Huang et al., 2025), harder detection of backdoor attacks if the model is as overconfident in its normal outputs as it is for poisoned outputs (Abbasi et al., 2023), and dangerous actions in health or control domains.

Bayesian Neural Networks (BNNs) (Mackay, 1995; Wilson & Izmailov, 2020; Charnock et al., 2020; Magris & Iosifidis, 2023) provide a powerful tool for uncertainty estimation by considering a distribution over the parameters of the neural network, but their stochastic structure commonly requires hundreds of model inferences to provide uncertainty estimation of good quality. The model samples can be applied in parallel, but when models are large (e.g., a single model inference requires the use of one GPU), then hundreds of GPUs are needed to achieve the same latency as a classical neural network, which is unrealistic for almost all applications.

In this work, motivated by the ranking process of Layer Ensembles (Oleksiienko & Iosifidis, 2023), we propose a method for improving the uncertainty quality of BNNs by reducing the continuous distribution to a set of samples and ranking these samples on a validation dataset to find the best subset for every number of samples used. Furthermore, following the limitations of real-world datasets, we expand this approach to the case where the validation dataset does not include the true uncertainty, as required by Layer Ensembles, but only provides its samples. We derive the algorithms for both cases and empirically show that the proposed methods can improve both the quality of uncertainty and the computational complexity of BNNs. We make the following contributions:

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

- We propose a method to improve the quality of uncertainty and computational requirements of Bayesian Neural Networks by ranking on a validation dataset the approximations of the posterior predictive distribution to closer match the true predictive distribution.
- We adapt the proposed method to datasets that cannot provide uncertainty by ranking based on log-likelihood on the validation dataset.
- We show the effectiveness of the proposed algorithms on a synthetic dataset that defines the true predictive distribution for the test and validation datasets.
- We compare the effectiveness of different ranking optimizations on uncertainty-labeled and regular datasets.

2 Related Work

Uncertainty in neural networks can be estimated by using deterministic methods (Sensoy et al., 2018; Zhong et al., 2020) that use a single model to predict both the output and the uncertainty, by introducing stochasticity in inputs (Wang et al., 2018; 2019; Kandel & Castelli, 2021), or by considering a distribution over the parameters of the neural network, resulting in BNNs. Depending on the choice of a probability distribution function over the parameters of the model, BNNs can be implemented in different forms, including Monte Carlo Dropout (MCD) (Gal & Ghahramani, 2016), Bayes By Backpropagation (BBB) (Blundell et al., 2015), Deep Ensembles (Osband et al., 2018), Hypermodels (Dwaracherla et al., 2020), Variational Neural Networks (VNNs) (Oleksiienko et al., 2023), and Layer Ensembles (Oleksiienko & Iosifidis, 2023).

BNNs estimate uncertainty by applying Monte Carlo integration to the full predictive distribution, which is implemented as computing the mean and the variance of multiple model inferences. Monte Carlo Dropout (Gal & Ghahramani, 2016) considers a Bernoulli distribution over the weights of the model. By applying dropout during the test phase, the model becomes stochastic. This method does not require any additional model parameters, making it an attractive add-on for already trained models. Bayes By Backpropagation (Blundell et al., 2015) considers a Gaussian distribution over the parameters of the model and utilizes Variational Inference (Hoffman et al., 2013) to find the Gaussian approximation of the posterior distribution. Each parameter in the network requires two values to be stored, i.e., the mean and the variance of the parameter, doubling the memory requirements. Deep Ensembles (Osband et al., 2018) consider a categorical distribution over the model's parameters and compute the complete predictive distribution as the mean and the variance of the outputs of all model choices. They are easy to train, but they require a lot of additional memory. Hypermodels (Dwaracherla et al., 2020) build a hypernetwork that takes as input a sample from a unit Gaussian distribution and learns to create parameters for a base network. The outputs from multiple sampled network parameters are used to compute the prediction and its uncertainty. Since the hypertorso needs parameters to generate each of the base model's weights, the total memory consumption can increase by an order of magnitude for reasonable models. Variational Neural Networks (VNNs) (Oleksiienko et al., 2023) consider a distribution over the hyperparameters of the network while keeping the network's weights fixed. VNNs double the number of parameters to generate each layer's predictive distribution from mean and variance values. Layer Ensembles (Oleksiienko & Iosifidis, 2023) consider an independent categorical distribution over the weights of each layer, resulting in a much higher number of possible network samples than in Deep Ensembles, but the sampled models can contain identical subnetworks. Layer Ensembles have the same memory structure as Deep Ensembles and require multiple times more parameters than the original model.

BNNs use statistically superior approaches but suffer from high computational costs. Layer Ensembles show that by ranking the full set of possible samples, the model can outperform all other BNNs in uncertainty quality while also reducing the number of computations needed by using only the most informative samples for the problem. The utilization of BNNs in real-world tasks is usually limited to Monte Carlo Dropout since it is easy to add on an existing model. Deep Ensembles are also easy to use, but due to their high requirements both in terms of time and memory resources, MCD is adopted more widely. However, MCD achieves the lowest quality of uncertainty among the presented methods (Osband et al., 2023; Oleksiienko et al., 2023; Oleksiienko & Iosifidis, 2023), undermining its effectiveness in real-world tasks that require uncertainty estimation. But even with better uncertainty quality, one of the biggest blocks in adopting BNNs for real-world tasks is their time complexity. In the following sections, we show how the time complexity and the uncertainty

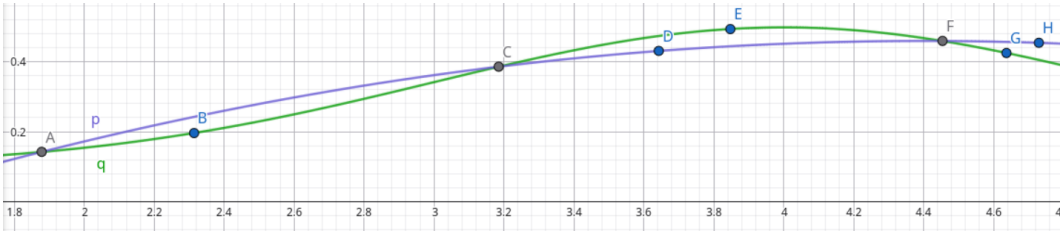


Figure 1: Two close predictive distributions that are approximated at different points

quality of BNNs can be improved by discretizing their continuous distributions and ranking the created sample subsets.

3 METHODOLOGY

BNN inference requires computing the integral of model-specific predictive distributions $p(y|x)$ over the network weights and hyperparameters as follows:

$$p(y|x) = \int \int p(y|x, w, a)\, p(w|a)\, p(a)\, da\, dw, \tag{1}$$

where $p(a)$ is a distribution over the model hyperparameters, $p(w|a)$ is a distribution over the model weights, and $p(y|x, w, a)$ is a model-specific predictive distribution of a neural network with fixed weights w and hyperparameters a applied to the input x. We can simplify the form of the integral by considering both weights and hyperparameters as a single variable ξ as follows:

$$p(y|x) = \int p(y|x, \xi) p(\xi)\, d\xi. \tag{2}$$

Since direct computation of this integral is computationally infeasible, we can approximate its value by applying importance sampling Monte Carlo integration as follows:

$$p(y|x) \approx \frac{1}{T} \sum_{i=1}^{T} p(y|x, \xi_i), \quad \xi_i \sim p(\xi), \tag{3}$$

where T is the number of samples used to approximate the integral. Higher values of T are expected to provide closer approximations of the integral.

Consider two BNNs applied on the same data point, predictive distributions of which are approximated following Equation 3, i.e.,:

$$\begin{aligned} p(y|x) = \int p(y|x, \xi) p(\xi)\, d\xi \quad &\approx \quad P(y|x) = \frac{1}{T_p} \sum_{i=1}^{T_p} p(y|x, \xi_i), \quad \xi_i \sim p(\xi), \\ q(y|x) = \int q(y|x, \eta) p(\eta)\, d\eta \quad &\approx \quad Q(y|x) = \frac{1}{T_q} \sum_{i=1}^{T_q} q(y|x, \eta_i), \quad \eta_i \sim p(\eta), \end{aligned} \tag{4}$$

where $P(y|x)$ and $Q(y|x)$ are the approximated values of the predictive distributions for the two selected BNNs created from T_p and T_q samples of parameters, respectively. For sufficiently close distributions $p(y|x)$ and $q(y|x)$, there exist samples of $p(\xi)$ and $p(\eta)$ such that $P(y|x) = Q(y|x)$, as can be seen in Figure 1.

In this example, the distribution $p(y|x, \xi)$ is approximated by points (A, B, C, D, F, H), and the distribution $q(y|x, \eta)$ is approximated by points (A, C, E, F, G). Considering the fact that any subset of the selected points for each distribution can be used for approximation, we can use points (A, C, F) to approximate both distributions when $T_p = T_q = 3$.

If we consider the distribution $q(y|x)$ to be the correct distribution for the selected problem, then finding such samples of ξ where the approximation of $p(y|x)$ is close to the approximation of $q(y|x)$ will result in a better solution of the problem than finding the exact value of $p(y|x)$. However, we need to be able to tell which samples of ξ drive the approximation of $p(y|x)$ closer to the true distribution $q(y|x)$.

3.1 DISCRETE BAYESIAN NEURAL NETWORKS

In this section we first define the ideal case where the true distribution can be accessed for a subset of the dataset and, after proposing the BNN discretization mechanism for this case, we expand the definition for real-world datasets that have only samples from the true distribution.

Consider the problem of finding a true distribution $q(y|x)$ given a training dataset $D = \{(x_t, y_t)\}$ that consists of inputs $x_t \sim p(x)$ and corresponding outputs $y_t \sim q(y|x)$, and a validation dataset $D_v = \{(x_*, q(y_*|x_*))\}$ that contains a true distribution $q(y_*|x_*)$ for each input $x_* \sim p(x)$. Let us define the approximation of $p(y|x)$ as

$$P(y|x, \Xi_k) = \frac{1}{k} \sum_{i=1}^{k} p(y|x, \xi_i), \quad \xi_i \in \Xi_k, \tag{5}$$

where Ξ_k is a set of k parameter samples drawn from $p(\xi)$. Given a "full" set of parameter samples Ξ_T that we consider for ranking, we want to find a subset Ξ_k for each value of $k \in \{1, ..., T\}$ that will provide the best approximation of $q(y|x)$ for k choices from Ξ_T. With a sufficiently high value of T, the approximation $P(y|x, \Xi_T)$ will be closer to the approximated distribution $p(y|x)$ than to the true distribution $q(y|x)$ if those distributions are different, which means that the full set Ξ_T should be a worse approximation of $q(y|x)$ than the best subset $\Xi_N, N < T$. We want to find such a value of N, and a corresponding subset Ξ_N, that provides the best estimate of $q(y|x)$ with the use of the validation dataset D_v. This process was shown to empirically work for Layer Ensembles (Oleksiienko & Iosifidis, 2023). We formally derive the underlying mechanisms and expand it for all types of BNNs. Additionally, we formulate a solution for datasets where the true predictive distribution cannot be accessed for any data point, which bridges the gap between synthetic tasks used by existing solutions and real-world problems.

The BNN is first trained on the dataset D to find the posterior predictive distribution $p(y|x, D)$ that is approximated by the set of posterior samples $\Xi_T = \{\xi_i\}$, $\xi_i \sim p(\xi|D)$. Then, we find the best sample subset Ξ_k for each subset size k as follows:

$$\Xi_k = \arg \min_{\Delta_k \subset \Xi_T} \mathrm{KL}(q(y|x)||P(y|x, \Delta_k)), \tag{6}$$

where $\mathrm{KL}(\cdot||\cdot)$ is the Kullback-Leibler divergence and Δ_k is any subset of Ξ_T containing k elements. Computing the KL-divergence requires evaluating the following double integral:

$$\mathbb{E}_{x \sim p(x)} \Big[\mathrm{KL}\big(q(y|x)||P(y|x, \Delta_k)\big) \Big] = \int \int q(y|x) \log \frac{q(y|x)}{P(y|x, \Delta_k)} p(x)\, dy\, dx, \tag{7}$$

which can be approximated by applying Monte Carlo integration with input samples from the validation dataset D_v as follows:

$$\int \int q(y|x) \log \frac{q(y|x)}{P(y|x, \Delta_k)} p(x)\, dy\, dx \approx \frac{1}{|D_v|} \sum_{x_* \in D_v} \int q(y_*|x_*) \log \frac{q(y_*|x_*)}{P(y_*|x_*, \Delta_k)}\, dy_*. \tag{8}$$

Since for each value of $x_* \in D_v$ the true distribution $q(y_*|x_*)$ can be obtained, we can approximate the total KL-divergence as a mean of KL-divergence values of distributions for each data point x_*.

To find the values of Ξ_k, we need to evaluate each subset Δ_k, resulting in $k\binom{T}{k}$ model inferences, which quickly becomes too computationally costly. Instead of directly finding the best subset Ξ_k, we can employ a greedy algorithm that starts with the direct computation of Ξ_1 and builds Ξ_{k+1} by finding the single best addition to Ξ_k as follows (Oleksiienko & Iosifidis, 2023; Caruana et al., 2004):

$$\begin{aligned} \Xi_1 &= \arg \min_{\Delta_1 \subset \Xi_T} \mathrm{KL}\big(q(y|x)||P(y|x, \Delta_1)\big), \\ \forall k, \Xi_{k+1} &= \Xi_k \cup \left\{ \arg \min_{\xi \notin \Xi_k, \xi \in \Xi_T} \mathrm{KL}\Big(q(y|x)||P\big(y|x, \Xi_k \cup \{\xi\}\big)\Big) \right\}. \end{aligned} \tag{9}$$

Such iterative algorithm greatly reduces the total number of model inferences needed while providing a good estimate of the best subset. If the computational budget allows, the full search can be used. Layer Ensembles apply each set of samples from scratch, producing unnecessary model

Algorithm 1 KL-divergence based ranking of BNNs

Input: Uncertainty dataset $D_v = \{(x_*, q(y_*|x_*))\}$, BNN model $F(x, \xi)$, maximum number of samples T, full sample set Ξ_T.
Initialize $\Delta = \varnothing$.
$\forall x_* \in D_v, \forall \xi \in \Xi_T$, compute $\Phi[x_*, \xi] = F(x_*, \xi)$.
for $k = 1$ **to** T **do**
  Set $\hat{\text{kl}} = \infty$ and $\hat{\xi} = \varnothing$.
  for $\xi \in \Xi_T$ and $\xi \notin \Delta$ **do**
    Set $\text{kl} = 0$.
    Set $\Delta_k = \Delta \cup \{\xi\}$.
    for $x_* \in D_v$ **do**
      Compute $P(y_*|x_*, \Delta_k) = \frac{1}{k} \sum_{\delta \in \Delta_k} \Phi[x_*, \delta]$.
      Set $\text{kl} = \text{kl} + \text{KL}(q(y_*|x_*)||P(y_*|x_*, \Delta_k))$.
    end for
    if $\text{kl} < \hat{\text{kl}}$ **then**
      Set $\hat{\text{kl}} = \text{kl}$, and $\hat{\xi} = \xi$.
    end if
  end for
  Set $\Delta = \Delta \cup \{\hat{\xi}\}$ and $\Xi_k = \Delta$.
end for
return $\Xi = \{\Xi_1, \Xi_2, ..., \Xi_T\}$.

inferences. We optimize this process by applying all T samples of the BNN to the whole validation dataset separately and storing the outputs. To evaluate a specific sample subset, we simply need to collect the stored outputs that correspond to the members of the subset and combine them, resulting in the probability distribution $P(y|x, \Delta_k)$ without the need to recompute the model outputs many times. Algorithm 1 implements KL-divergence based ranking with the aforementioned optimizations by storing model inferences in a Φ array and iteratively building the best sets Ξ_k for all values of k. The best subset Ξ_N is selected as a subset with the lowest KL-divergence value among the ranked subsets.

The memory use of the Φ array is negligible for regular classification and regression tasks that produce low-dimensional outputs but can become overwhelming if the task requires producing outputs of the same size (such as segmentation) or bigger (such as up-scaling) than the input data. If the Φ array does not fit to the memory, it can be stored on disk, and the algorithm can load only the parts of it relevant to the inputs being processed. Continuous BNNs, such as BBB, VNNs, Hypermodels, and MCD, can use any limit of samples T for ranking, while already discrete models such as Deep Ensembles and Layer Ensembles can simply use the maximum number of possible samples. However, if the number of possible samples is too big, we can limit the search by selecting T to be smaller than the maximum number of samples in the model and ranking those randomly selected samples. This can be particularly helpful for deep Layer Ensembles that can reach millions of combinations. The requirement of validation dataset uncertainty can be satisfied by either using a full dataset with uncertainty, such as MUSES (Brödermann et al., 2024), CIFAR-10H (Peterson et al., 2019), CIFAR-10S (Collins et al., 2022), or creating uncertainty targets only for a small validation subset, which greatly reduces the required workload. Alternatively, the ranking process can be done on common real-world datasets by approximating KL-divergence ranking with log-likelihood ranking.

3.2 Application to real-world datasets

When the validation dataset D_v contains only samples y_* from the true distribution $q(y_*|x_*)$ instead of the whole distribution, as is common in real-world datasets, we can still use KL-divergence to

rank the subsets Δ_k but we need to approximate it with only one sample as follows:

$$\begin{aligned}\mathrm{KL}\big(q(y|x)||P(y|x,\Delta_k)\big) &= \int\int q(y|x)\log\frac{q(y|x)}{P(y|x,\Delta_k)}\,dy\,dx \\ &\approx \frac{1}{|D_v|}\sum_{x_*\in D_v}\int q(y_*|x_*)\log\frac{q(y_*|x_*)}{P(y_*|x_*,\Delta_k)}\,dy_* \\ &\approx \frac{1}{|D_v|}\sum_{(x_*,y_*)\in D_v}\log\frac{q(y_*|x_*)}{P(y_*|x_*,\Delta_k)}.\end{aligned} \tag{10}$$

We still require computing the unknown distribution $q(y_*|x_*)$ to find the KL-divergence. Instead of using the exact KL-divergence values to rank the subsets, we can compare two subsets Δ_k and $\bar{\Delta}_k$ by computing the difference between the two corresponding KL-divergence values. If the difference is positive, then $\bar{\Delta}_k$ should be favored over Δ_k. The difference of KL-divergence values is computed as follows:

$$\begin{aligned}&\mathrm{KL}(q(y|x)||P(y|x,\Delta_k)) - \mathrm{KL}(q(y|x)||P(y|x,\bar{\Delta}_k)) \approx \\ &\approx \frac{1}{|D_v|}\sum_{(x_*,y_*)\in D_v}\log\frac{q(y_*|x_*)}{P(y_*|x_*,\Delta_k)} - \frac{1}{|D_v|}\sum_{(x_*,y_*)\in D_v}\log\frac{q(y_*|x_*)}{P(y_*|x_*,\bar{\Delta}_k)} \\ &= \frac{-1}{|D_v|}\sum_{(x_*,y_*)\in D_v}\log P(y_*|x_*,\Delta_k) + \frac{1}{|D_v|}\sum_{(x_*,y_*)\in D_v}\log P(y_*|x_*,\bar{\Delta}_k).\end{aligned} \tag{11}$$

The difference of KL-divergence values will be positive if the difference of log-likelihoods of the validation dataset for selected subsets Δ_k and $\bar{\Delta}_k$ is negative. This means that, if the true distributions are not provided for the validation dataset, we can rank models based on their log-likelihood values on the validation dataset as

$$\begin{aligned}\Xi_k &= \arg\min_{\Delta_k\subset\Xi_T}\mathrm{KL}(q(y|x)||P(y|x,\Delta_k)) \\ &\approx \arg\max_{\Delta_k\subset\Xi_T}\sum_{(x_*,y_*)\in D_v}\log P(y_*|x_*,\Delta_k).\end{aligned} \tag{12}$$

Appendix A shows the log-likelihood ranking algorithm of BNNs that follows the same optimizations as for KL-divergence ranking but, instead of searching for the minimum of the KL-divergence, it computes log-likelihoods of the dataset for each sample set and finds the set with the highest value. As a result of the ranking process, we can select the best sample sets for each number of samples in the range $[1, ..., T]$. We call each model obtained by using the set Ξ_k a *Discrete Bayesian Neural Network (DiBNN)* created from the base BNN. DiBNN models are not stochastic and are obtained by discretization and ranking of the usually continuous distribution function over its weights and hyperparameters.

4 Experiments

The Neural Testbed (Osband et al., 2022) implements a problem of predicting the output distribution for a set of data points with properties presented in Section 3.1 to estimate the quality of the predictive distribution by generating a training dataset $D = \{(x_t, y_t)\}$, where $x_t \sim \mathcal{N}(x|0, I)$, $y_t \sim q(y|x_t)$, and the true distribution $q(y|x_t)$ is obtained as the result of a Neural Network Gaussian Process (NNGP) as follows:

$$\begin{aligned}q(y|x) &= \mathcal{N}(\mu, \Sigma), \\ (\mu, \Sigma) &= \mathrm{NNGP}(x),\end{aligned} \tag{13}$$

with $\mathrm{NNGP}(\cdot)$ being the Neural Network Gaussian Process function. A test dataset D_t is created containing elements $(x_\dagger, q(y_\dagger|x_\dagger))$ that are used to estimate KL-divergence between the evaluated BNN predictive distribution and the true distribution from the NNGP model. Layer Ensembles introduce a validation dataset $D_v = \{(x_*, q(y_*|x_*))\}$, sampled in the same way as the test dataset, that is used for finding the best samples of DiBNNs in this work. Additionally, we use a second

Table 1: Comparison of KL-divergence based ranked DiBNN models and BNNs. KL-divergence and sample metrics are presented as a "median (interquartile range)" value over 27 datasets with different parameters. "DiBNN KL" is a KL-divergence value of the ranked DiBNN model. "DiBNN Best KL" is the best KL-divergence possible that can be achieved by using the ranking process. "DiBNN" and "DiBNN Best" samples describe how many samples were used to achieve the corresponding results. "Failure" % describes how many models resulted in KL-divergence higher than 5.0. "Max Samples" shows the value of T considered for the experiment. BNN results at N or T samples show how a random BNN inference compares to the model with the same samples as the DiBNN model or as all T samples, accordingly.

Method	Ensemble	VNN	Hypermodel	Layer Ensembles	BBB	MCD
DiBNN KL ↓	0.10 (0.12)	0.17 (0.28)	0.19 (0.51)	0.27 (0.16)	0.40 (0.52)	0.59 (1.51)
DiBNN Best KL ↓	0.08 (0.09)	0.16 (0.28)	0.19 (0.51)	0.23 (0.14)	0.40 (0.48)	0.59 (1.32)
DiBNN Samples (N) ↓	19.0 (7.5)	64.0 (80.0)	60.0 (40.0)	14.0 (7.0)	19.0 (43.0)	44.0 (48.5)
DiBNN Best Samples ↓	27.0 (8.5)	150.0 (192.0)	106.0 (175.5)	18.0 (11.0)	35.0 (767.0)	964.0 (398.5)
Max Samples (T)	30	1000	1000	27	1000	1000
Failure % ↓	0.0%	0.0%	0.0%	0.0%	0.0%	11.1%
BNN KL at N Samples ↓	0.15 (0.11)	0.45 (0.92)	0.49 (0.71)	0.28 (0.17)	0.52 (1.14)	0.78 (2.45)
BNN KL at T Samples ↓	0.11 (0.09)	0.36 (0.35)	0.35 (0.61)	0.25 (0.10)	0.48 (0.61)	0.59 (1.32)

Table 2: Comparison of log-likelihood-ranked DiBNN models and BNNs. The table is structured identically to Table 1.

Method	Ensemble	Layer Ensembles	MCD	VNN	BBB	Hypermodel
DiBNN KL ↓	0.30 (11.51)	0.40 (0.54)	0.66 (31.41)	0.70 (9.18)	1.02 (1.55)	1.25 (4.10)
DiBNN Best KL ↓	0.10 (0.09)	0.23 (0.11)	0.59 (1.33)	0.31 (0.33)	0.40 (0.66)	0.32 (0.59)
DiBNN Samples (N) ↓	7.0 (11.0)	6.0 (9.0)	10.0 (30.0)	8.0 (16.0)	7.0 (13.0)	9.0 (27.0)
DiBNN Best Samples ↓	30.0 (3.0)	25.0 (5.0)	798.0 (293.5)	640.0 (724.5)	923.0 (450.0)	826.0 (807.0)
Max Samples (T)	30	27	1000	1000	1000	1000
Failure % ↓	33.3%	14.8%	33.3%	29.6%	14.8%	25.9%
BNN KL at N Samples ↓	0.34 (16.31)	0.41 (0.38)	1.97 (7.85)	1.88 (7.49)	1.41 (5.99)	1.63 (53.72)
BNN KL at T Samples ↓	0.11 (0.09)	0.25 (0.10)	0.59 (1.32)	0.36 (0.35)	0.48 (0.61)	0.35 (0.61)

version of the validation dataset called the point-wise validation dataset $\bar{D}_v = \{(x_*, y_*)\}$, where the ground-truth values y_* are sampled from the corresponding distribution $q(y_*|x_*)$. The models are trained on the training dataset with early stopping based on either KL-divergence or log-likelihood of the model on the corresponding validation dataset. The best model hyperparameters are defined based on the validation KL-divergence, and only the best model for each BNN is used for creating the discrete version of the model. Such a process allows us to use the validation dataset during BNN training and lets us compare the discrete BNNs with regular BNNs optimized on the same data. We test the DiBNN models obtained by ranking on D_v and $\bar{D}_v$ separately and show the comparison of the test KL-divergence for each evaluated BNN type and number of samples used in the range $[1, ..., T]$. We translated the Neural Testbed experiments from JAX to PyTorch to make it more widely accessible and to improve the parallelization of the existing and added experiments[1].

Table 1 shows a comparison between KL-ranked DiBNN (Calibrated KL) and the original BNNs (Random Set KL) across 27 generated datasets with different numbers of input dimensions $(1, 10, 100)$, data ratio $(1, 10, 100)$ and noise scale $(0.01, 0.1, 1.0)$ with $T = 1000$ max samples for continuous BNNs, 30 samples for the Deep Ensembles model with 30 members, and 27 samples for the Layer Ensembles model with 3 members. Every tested DiBNN has achieved a lower KL-divergence score than the corresponding BNN at the same number of samples, and Deep Ensemble, VNN, Hypermodel and BBB achieved a better KL-divergence value than the corresponding BNN model at max samples. The resulting DiBNNs require much fewer samples for inference to achieve those results, and due to the nature of building those samples, each prefix of the sample set can be used as the best set for an application where the computational demand cannot be fully met.

[1]The code is available at `https://anonymous.4open.science/r/dibnn-CE33`

Table 3: Comparison of log-likelihood-ranked discrete VNN and BBB models and random-sampled VNN and BBB models at the same number of samples with $T = 100$ max samples. "DiVNN KL" represents the discrete VNN KL-divergence (interquartile range), and "VNN KL" represents the same values for regular VNNs. "DiBBB KL" and "BBB KL" represent the corresponding discrete BBB and regular BBB values.

Samples	DiVNN KL	VNN KL	DiBBB KL	BBB KL
2	107.67 (145.81)	159.92 (93.30)	1.16 (1.74)	2.75 (3.25)
5	0.94 (1.24)	1.19 (2.48)	1.02 (1.45)	0.58 (0.55)
7	0.64 (1.35)	0.66 (0.60)	0.97 (1.54)	0.57 (0.60)
10	0.55 (1.49)	0.72 (1.02)	0.95 (1.54)	0.54 (0.61)
12	0.53 (1.11)	0.53 (0.84)	0.93 (1.55)	0.61 (0.54)
15	0.45 (1.22)	0.48 (0.47)	0.89 (1.62)	0.56 (0.62)
17	0.43 (1.04)	0.44 (0.49)	0.86 (1.63)	0.50 (0.61)
20	0.42 (1.14)	0.45 (0.47)	0.84 (1.65)	0.55 (0.61)
22	0.41 (1.19)	0.48 (0.50)	0.81 (1.67)	0.52 (0.61)
25	0.40 (1.32)	0.45 (0.48)	0.79 (1.69)	0.51 (0.61)
32	0.39 (0.93)	0.40 (0.39)	0.70 (1.70)	0.51 (0.60)
40	0.38 (0.89)	0.41 (0.43)	0.65 (1.37)	0.50 (0.62)
47	0.38 (0.72)	0.39 (0.42)	0.62 (0.80)	0.52 (0.61)
55	0.37 (0.71)	0.39 (0.42)	0.59 (0.72)	0.49 (0.61)
62	0.35 (0.65)	0.38 (0.40)	0.53 (0.62)	0.50 (0.60)
70	0.36 (0.56)	0.38 (0.36)	0.51 (0.67)	0.49 (0.62)
77	0.37 (0.50)	0.38 (0.38)	0.48 (0.72)	0.48 (0.62)
85	0.37 (0.44)	0.34 (0.36)	0.45 (0.75)	0.48 (0.61)
92	0.37 (0.39)	0.38 (0.43)	0.49 (0.74)	0.51 (0.60)
100	0.34 (0.33)	0.37 (0.39)	0.50 (0.65)	0.48 (0.61)

Real-world datasets experimental setup Table 2 shows the results of the log-likelihood-ranking scheme applied under the common real-world use case when uncertainty targets are not available for the validation dataset. Due to limitations of log-likelihood approximation of comparison between distributions, none of the resulting DiBNNs could outperform the corresponding BNN at max samples (T), but if we compare the models with the same number of samples used (N), DiBNN models still produce better results than BNNs. The LL-based ranking stops early for all tested methods and achieves multiple times higher test KL-divergence, resulting in median DiBNN samples of 10 or lower, while the best samples are close to the max sample count. This indicates that the LL-based ranking of the models, while being less accurate than the KL-divergence-based ranking, can still be used to improve the models' performance, especially for limited computational budgets. Table 3 shows the comparison between LL-ranked discrete and regular VNN and BBB models at fixed samples for both with $T = 100$. Given a limited computational budget, discrete VNNs will achieve a better quality of uncertainty compared to randomly sampled VNNs while the improvement for discrete BBBs can only be seen at higher numbers of samples. As shown in Appendix B, Layer Ensembles, similarly to BBBs, show an improvement closer to the max samples and a degradation of KL-divergence for smaller numbers of samples, while Hypermodels show an improvement for smaller numbers of samples. Monte Carlo Dropout behaves differently from other BNNs and shows almost no difference between ranked and random samples.

Comparison with other ranking processes We follow the optimizations proposed by Caruana et al. (2004) and introduce: 1) the initialization process that selects first L samples by looking for samples with the best single-sample performance on the validation dataset and only then continues with the ranking process, and 2) the ranking process that allows for duplicate samples to appear in the same set. Table 4 compares the results of the initialization process with 5 (Init 5) and 15 (Init 15) samples, the replacement ranking (Replace), and the regular ranking (Init 0) of VNNs with $T = 100$ max samples. While ranking with replacement, we consider a maximum number of $2T$ samples to allow for a wider search of repeating samples. When the uncertainty targets are given for the validation dataset, all ranking processes result in almost identical DiBNN KL-divergence values, with the replacement having the worst results and higher variance among them while also using almost $2T$ samples with many repetitive samples being selected. The initialization with 5

Table 4: Comparison of regular ranking (Init 0), ranking with initialization (Init 5/15), and ranking with replacement (Replace). Ranking is done with KL-divergence if "KL" is present in the method name and with log-likelihood otherwise.

Method	Init 0 KL	Init 5 KL	Init 15 KL	Replace KL	Init 0 LL	Init 5 LL	Init 15 LL	Replace LL
DiVNN KL ↓	0.26 (0.26)	0.25 (0.26)	0.26 (0.26)	0.28 (0.41)	0.64 (12.82)	0.93 (7.16)	0.67 (2.53)	0.72 (3.75)
DiVNN Best KL ↓	0.22 (0.27)	0.22 (0.27)	0.22 (0.27)	0.24 (0.32)	0.29 (0.32)	0.29 (0.32)	0.29 (0.33)	0.58 (0.89)
DiVNN Samples (N) ↓	19.0 (16.5)	24.0 (16.0)	19.0 (16.5)	191.0 (24.5)	5.0 (9.0)	6.0 (5.0)	20.0 (27.0)	200.0 (0.0)
DiVNN Best Samples ↓	44.0 (38.0)	46.0 (30.0)	44.0 (38.0)	44.0 (67.5)	77.0 (36.0)	78.0 (40.0)	78.0 (40.0)	16.0 (25.0)
Failure % ↓	7.4%	0.0%	7.4%	0.0%	33.3%	29.6%	22.2%	25.9%
VNN KL at N Samples ↓	0.59 (1.12)	0.46 (0.42)	0.48 (1.24)	0.36 (0.38)	1.80 (67.41)	0.77 (3.31)	0.67 (3.71)	0.36 (0.38)
VNN KL at T Samples ↓	0.36 (0.38)	0.36 (0.38)	0.36 (0.38)	0.36 (0.38)	0.36 (0.38)	0.36 (0.38)	0.36 (0.38)	0.36 (0.38)

models achieves a 0% failure rate, while initializing with 15 models and the regular ranking achieve a KL-divergence of 5 or higher in 7.4% of cases. However, it requires 5 more models on average to find the best set. If ranking is done with log-likelihood only, then initializing the models results in worse uncertainty quality than for the regular ranking. Ranking with replacement achieves the worst results and requires all $2T$ samples at all times.

5 Conclusions

In this paper, we proposed a method to discretize Bayesian Neural Networks and rank their samples to create deterministic models that achieve higher uncertainty quality and require fewer computations to do so. We formally derive the ranking process of Layer Ensembles and extend it to other BNNs and datasets that do not have uncertainty targets for the validation dataset and show the effectiveness of both ranking methods by applying them to Monte Carlo Dropout (Gal & Ghahramani, 2016), Hypermodels (Dwaracherla et al., 2020), Variational Neural Networks (Oleksiienko et al., 2023), Deep Ensembles (Osband et al., 2018), Layer Ensembles (Oleksiienko & Iosifidis, 2023) and Bayes By Backpropagation (Blundell et al., 2015). The constructed Discrete Bayesian Neural Networks achieve up to 2.1 times higher uncertainty quality with an order of magnitude lower computational requirements if a validation dataset uncertainty is given. For the regular datasets, we show how ranking can provide better models for systems with limited resources or dynamic resource allocation.

AI use statement

In this work, we used generative AI tools for assisting with software creation. We have not used generative AI tools to generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, support qualitative and thematic data analysis, interpret results, and tasks such as "assist in the writing of proofs", "assist with translation", "clean and reformat dataset" are not applicable to this work. We have reviewed all AI-assisted work. LLM-generated code parts were verified and tested for correctness both experimentally and theoretically. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

Reproducibility statement

The algorithms presented in the paper and the experimental protocol are implemented in PyTorch at `https://anonymous.4open.science/r/dibnn-CE33`. The implementation considers fixed seeds, and all generated random values are created by splitting the provided random seeds, which results in fully repeatable experiments as long as the experiment parameters are the same. We provide the tools needed to build tables and figures presented in the paper.

REFERENCES

Bilal Hussain Abbasi, Qi Zhong, Leo Yu Zhang, Shang Gao, Antonio Robles-Kelly, and Robin Doss. A generic enhancer for backdoor attacks on deep neural networks. In *Neural Information Processing*, volume 1794, pp. 296–307, 2023.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight Uncertainty in Neural Networks. In *International Conference on Machine Learning (ICML)*, volume 37, pp. 1613–1622, 2015.

Tim Brödermann, David Bruggemann, Christos Sakaridis, Kevin Ta, Odysseas Liagouris, Jason Corkill, and Luc Van Gool. Muses: The multi-sensor semantic perception dataset for driving under uncertainty. In *European Conference on Computer Vision (ECCV)*, volume 15117, pp. 21–38, 2024.

Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *International Conference on Machine Learning (ICML)*, volume 69, pp. 18, 2004.

Tom Charnock, Laurence Perreault-Levasseur, and François Lanusse. Bayesian neural networks. *arXiv:2006.01490*, 2020.

Katherine M Collins, Umang Bhatt, and Adrian Weller. Eliciting and learning with soft labels from every annotator. In *AAAI Conference on Human Computation and Crowdsourcing (HCOMP)*, volume 10, pp. 40–52, 2022.

Vikranth Dwaracherla, Xiuyuan Lu, Morteza Ibrahimi, Ian Osband, Zheng Wen, and Benjamin Van Roy. Hypermodels for exploration. In *International Conference on Learning Representations (ICLR)*, volume 8, 2020.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning (ICML)*, volume 48, pp. 1050–1059, 2016.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning (ICML)*, volume 70, pp. 1321–1330, 2017.

Matt Hoffman, David M. Blei, Chong Wang, and John Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):42:1–42:55, 2025.

Ibrahem Kandel and Mauro Castelli. Improving convolutional neural networks performance for image classification using test time augmentation: a case study using MURA dataset. *Health Information Science and Systems*, 9(1):33, 2021.

David J C Mackay. Probable networks and plausible predictions — a review of practical bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6(3):469–505, 1995.

Martin Magris and Alexandros Iosifidis. Bayesian learning for neural networks: an algorithmic survey. *Artificial Intelligence Review*, 56(10):11773–11823, 2023.

Illia Oleksiienko and Alexandros Iosifidis. Layer ensembles. *IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6, 2023.

Illia Oleksiienko, Dat Thanh Tran, and Alexandros Iosifidis. Variational neural networks. *Procedia Computer Science*, 222:104–113, 2023.

Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. In *Advances on Neural Information Processing Systems (NeurIPS)*, volume 31, pp. 8626–8638, 2018.

Ian Osband, Zheng Wen, Seyed Mohammad Asghari, Vikranth Dwaracherla, Xiuyuan Lu, Morteza Ibrahimi, Dieterich Lawson, Botao Hao, Brendan O' Donoghue, and Benjamin Van Roy. The neural testbed: Evaluating joint predictions. In *Advances on Neural Information Processing Systems (NeurIPS)*, volume 35, pp. 12554–12565, 2022.

Ian Osband, Zheng Wen, Seyed Mohammad Asghari, Vikranth Dwaracherla, Morteza Ibrahimi, Xiuyuan Lu, and Benjamin Van Roy. Epistemic neural networks. In *Advances on Neural Information Processing Systems (NeurIPS)*, volume 36, pp. 2795–2823, 2023.

Joshua C. Peterson, Ruairidh M. Battleday, Thomas L. Griffiths, and Olga Russakovsky. Human uncertainty makes classification more robust. In *IEEE International Conference on Computer Vision (ICCV)*, pp. 9616–9625, 2019.

Murat Sensoy, Lance Kaplan, and Melih Kandemir. Evidential deep learning to quantify classification uncertainty. In *Advances on Neural Information Processing Systems (NeurIPS)*, volume 31, 2018.

Guotai Wang, Wenqi Li, Sébastien Ourselin, and Tom Vercauteren. Automatic brain tumor segmentation using convolutional neural networks with test-time augmentation. In *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*, volume 11384, pp. 61–72, 2018.

Guotai Wang, Wenqi Li, Michael Aertsen, Jan Deprest, Sébastien Ourselin, and Tom Vercauteren. Aleatoric uncertainty estimation with test-time augmentation for medical image segmentation with convolutional neural networks. *Neurocomputing*, 338:34–45, 2019.

Andrew G Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. In *Advances on Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 4697–4708, 2020.

Yuanxin Zhong, Minghan Zhu, and Huei Peng. Uncertainty-aware voxel based 3d object detection and tracking with von-mises loss. *arXiv:2011.02553*, 2020.

A LOG-LIKELIHOOD RANKING ALGORITHM

Algorithm 2 presents the process of ranking based on log-likelihood scores on the validation dataset. The outputs of the BNN model for each input and sample are cached in the Φ array and are later reused for fast ranking of the sample sets. The method follows the same idea as the KL-divergence-based ranking, where the best sets are iteratively generated by adding a new sample to the previous best set to achieve the highest log-likelihood values.

Algorithm 2 Log-likelihood ranking of BNNs

Input: Dataset $D_v = \{(x_*, y_*)\}$, BNN model $F(x, \xi)$, maximum number of samples T, full sample set Ξ_T.
Initialize $\Delta = \varnothing$.
$\forall x_* \in D_v, \forall \xi \in \Xi_T$, compute $\Phi[x_*, \xi] = F(x_*, \xi)$.
for $k = 1$ **to** T **do**
  Set $\hat{\text{ll}} = -\infty$ and $\hat{\xi} = \varnothing$.
  for $\xi \in \Xi_T$ and $\xi \notin \Delta$ **do**
    Set $\text{ll} = 0$.
    Set $\Delta_k = \Delta \cup \{\xi\}$.
    for $x_* \in D_v$ **do**
      Compute $P(y_*|x_*, \Delta_k) = \frac{1}{k} \sum_{\delta \in \Delta_k} \Phi[x_*, \delta]$.
      Set $\text{ll} = \text{ll} + \log P(y_*|x_*, \Delta_k)$.
    end for
    if $\text{ll} > \hat{\text{ll}}$ **then**
      Set $\hat{\text{ll}} = \text{ll}$ and $\hat{\xi} = \xi$.
    end if
  end for
  Set $\Delta = \Delta \cup \{\hat{\xi}\}$ and $\Xi_k = \Delta$.
end for
return $\Xi = \{\Xi_1, \Xi_2, ..., \Xi_T\}$.

B LOG-LIKELIHOOD RANKING OF HYPERMODEL, MCD, ENSEMBLE AND LAYER ENSEMBLE MODELS BY SAMPLES

Tables 5, 6 and 7 show the comparison of DiBNN and BNN versions of Hypermodels, MCD, Deep Ensembles, and Layer Ensembles when the same number of samples is used for all datasets with max samples $T = 100$ for continuous models. The DiMCD and MCD models achieve almost identical performance for higher numbers of samples and have slight variations with smaller samples. The DiBNN versions of Hypermodels, Deep Ensembles, and Layer Ensembles achieve better KL values for small numbers of samples, which indicates that those models can be used to improve performance under limited computational resources.

Table 5: Comparison of log-likelihood-ranked discrete Hypermodel and MCD models and random-sampled Hypermodel and MCD models at the same number of samples with $T = 100$ max samples. "DiHypermodel KL" represents the discrete VNN KL-divergence (interquartile range), and "Hypermodel KL" represents the same values for regular Hypermodels. "DiMCD KL" and "MCD KL" represent the corresponding discrete MCD and regular MCD values.

Samples	DiHypermodel KL	Hypermodel KL	DiMCD KL	MCD KL
2	51.51 (126.25)	196.08 (155.02)	187.20 (193.95)	204.79 (197.49)
5	1.13 (0.99)	2.14 (1.64)	1.45 (1.75)	1.32 (1.76)
7	0.99 (0.70)	0.82 (0.78)	0.98 (1.67)	1.02 (1.64)
10	0.77 (0.63)	1.03 (0.84)	0.84 (1.49)	0.86 (1.50)
12	0.76 (0.58)	0.80 (0.76)	0.80 (1.46)	0.80 (1.51)
15	0.60 (0.64)	0.51 (0.67)	0.75 (1.43)	0.76 (1.44)
17	0.60 (0.69)	0.61 (0.71)	0.73 (1.42)	0.74 (1.43)
20	0.56 (0.55)	0.50 (0.61)	0.71 (1.40)	0.71 (1.44)
22	0.58 (0.52)	0.51 (0.75)	0.70 (1.38)	0.71 (1.42)
25	0.56 (0.50)	0.45 (0.67)	0.68 (1.39)	0.68 (1.39)
32	0.59 (0.53)	0.43 (0.59)	0.66 (1.39)	0.67 (1.38)
40	0.48 (0.53)	0.42 (0.61)	0.65 (1.37)	0.65 (1.37)
47	0.49 (0.51)	0.43 (0.61)	0.64 (1.36)	0.63 (1.36)
55	0.47 (0.52)	0.39 (0.62)	0.63 (1.35)	0.63 (1.36)
62	0.47 (0.50)	0.37 (0.59)	0.62 (1.34)	0.62 (1.33)
70	0.43 (0.52)	0.38 (0.62)	0.62 (1.34)	0.62 (1.34)
77	0.39 (0.53)	0.37 (0.61)	0.62 (1.35)	0.61 (1.35)
85	0.41 (0.59)	0.37 (0.62)	0.61 (1.35)	0.61 (1.33)
92	0.41 (0.63)	0.38 (0.61)	0.61 (1.35)	0.61 (1.34)
100	0.41 (0.66)	0.35 (0.62)	0.61 (1.35)	0.61 (1.34)

Table 6: Comparison of log-likelihood-ranked Discrete Deep Ensemble models and random-sampled Deep Ensemble models at the same number of samples with $T = 30$ max samples. "DiEnsemble KL" represents the discrete Deep Ensemble KL-divergence (interquartile range), and "Ensemble KL" represents the same values for regular Ensembles.

Samples	DiEnsemble KL	Ensemble KL
2	102.09 (101.78)	102.78 (104.80)
3	3.61 (2.28)	3.90 (3.17)
4	0.88 (0.49)	0.89 (0.24)
5	0.53 (0.24)	0.52 (0.11)
6	0.39 (0.21)	0.39 (0.10)
7	0.33 (0.14)	0.31 (0.06)
9	0.26 (0.10)	0.23 (0.08)
11	0.21 (0.11)	0.18 (0.08)
13	0.19 (0.11)	0.19 (0.09)
16	0.17 (0.13)	0.13 (0.08)
18	0.16 (0.12)	0.12 (0.08)
20	0.16 (0.11)	0.13 (0.08)
23	0.15 (0.09)	0.11 (0.11)
25	0.13 (0.10)	0.11 (0.11)
27	0.12 (0.09)	0.12 (0.08)
30	0.11 (0.09)	0.11 (0.09)

Table 7: Comparison of log-likelihood-ranked Discrete Layer Ensembles models and random-sampled Layer Ensembles models at the same number of samples with $T = 27$ max samples. "DiLayerEnsembles KL" represents the discrete Layer Ensembles KL-divergence (interquartile range), and "Layer Ensembles KL" represents the same values for regular Layer Ensembles.

Samples	DiLayerEnsembles KL	Layer Ensembles KL
2	89.18 (101.46)	111.79 (98.64)
3	2.62 (2.12)	2.22 (1.05)
4	0.87 (0.27)	0.88 (0.23)
5	0.56 (0.17)	0.55 (0.19)
6	0.46 (0.17)	0.42 (0.13)
8	0.36 (0.15)	0.34 (0.13)
10	0.34 (0.16)	0.31 (0.10)
12	0.32 (0.15)	0.30 (0.10)
14	0.30 (0.15)	0.27 (0.09)
16	0.26 (0.10)	0.28 (0.10)
18	0.25 (0.10)	0.27 (0.09)
20	0.26 (0.10)	0.25 (0.09)
22	0.25 (0.10)	0.26 (0.10)
24	0.24 (0.10)	0.26 (0.10)
27	0.25 (0.10)	0.25 (0.10)

C RANKING CURVES OF HYPERMODELS

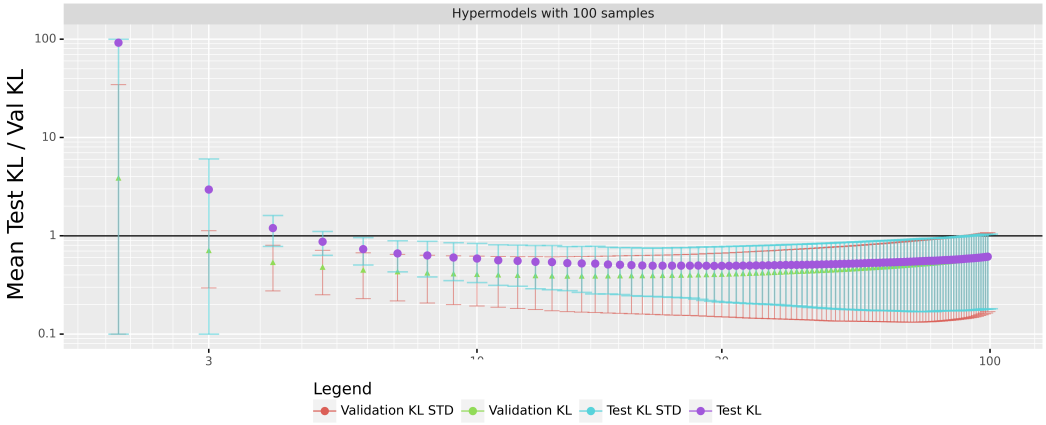


Figure 2: Mean and Std of test and validation KL-divergence values for each subset size of a Discrete Hypermodel with $T = 100$ max samples.

Figure 2 presents the mean test KL-divergence and variance of all ranked samples of Hypermodels with $T = 100$ max samples. The test KL-divergence of the best combination of samples for each number of samples is presented in the figures as a purple circle with a cyan error bar representing the variance of the scores based on dataset configurations, which include the dimensionality of the data, the number of data points, and the scale of the noise added to the data. The validation KL-divergence is represented as a green triangle with a red variance bar. The best points of both test and validation KL-divergence values are achieved at lower samples than T and the corresponding scores keep getting worse after the best sample sets are reached. Those best points are different due to a disparity between validation and test sets.