Guided Deep Kernel Learning - Supplementary Material

Idan Achituve[1] **Gal Chechik**[2, 3] **Ethan Fetaya**[1]

[1]Faculty of Engineering, Bar-Ilan University, Israel
[2]Computer Science Dept., Bar-Ilan University, Israel
[3]NVIDIA, Israel

A EXPERIMENTAL DETAILS

All experiments were done with GPyTorch [Gardner et al., 2018] on NVIDIA GeForce RTX 2080 Ti having 11GB of memory. To compute the kernel of the NNGP we used the Neural Tangents library [Novak et al., 2020].

Toy Dataset. To construct the toy example we define the following target function (following [Leclercq, 2018]): $f(x) = 0.6 - e^{-(x-2)^2} - e^{-\frac{1}{10}(x-6)^2} - \frac{1}{x^2+1}$. We sample uniformly at random 800 points in $[-2, 12]$, evaluate the function on them and add observation noise of $\sigma_n^2 = 0.05$. Then we partition the dataset by random sampling to two subsets of 400 points each, namely $\mathcal{D}_1$ and $\mathcal{D}_2$. We remove from $\mathcal{D}_1$ all the points from the domain $[4, 8]$, and fit a GP with an RBF kernel to this dataset. We learn the hyper-parameters of this kernel using the ADAM optimizer [Kingma and Ba, 2015] with the log marginal likelihood. Then, we evaluate $p(f_*|x_*, \mathcal{D}_1)$ and $p(f_*|x_*, y_*, \mathcal{D}_1)$ for all $(x_*, y_*) \in \mathcal{D}_2$.

UCI. We followed most of the training protocol suggested in [Ober et al., 2021]. To download and manipulate the datasets we used the Bayesian benchmarks git repository: [https://github.com/hughsalimbeni/bayesian_benchmarks]. To train the models, on Boston, Concrete, and Energy we perform 10-fold cross-validation using random seeds according to $90\% - 10\%$ train-test splits. On Buzz and CTSlice we perform 3-fold cross-validation. We computed the normalization statistics (i.e., mean and std) based on the train split only and normalize all the data using them for model fitting. However, the results shown in the paper are on the unnormalized target values (i.e., the original values) which differ only in the scale. In all experiments, we used a fully connected network with the following architecture $[d, 100, 100, 100, 20]$ and ReLU activations. We used the same number of layers and activation for the infinite-width network. We initialized the variance of the observation noise to ~ 0.02 and learned it along with the model parameters. We used a weight decay of $1e-4$ for the DKL model only (no weight decay for GDKL), and we set the variance of the weights and biases of the NNGP to 1.6 and 0.2 as we found these values to work well on several toy examples. We trained all baseline models for 8000 iterations. In GDKL we first pre-train the NNGP model observation noise and output scale of the kernel for 1000 iterations, and then we train the DKL model for another 7000 iterations in order to be comparable in the number of gradient steps used by the baseline methods. Also, on Buzz and CTSlice we used $\beta = 1.2$ as we found it to work slightly better than 1 on a predefined validation set. We used a learning rate of $1e-2$ which drops by a factor of 10 after 60% and 80% of the training (not including the pre-train stage of GDKL).

CIFAR-10/100. CIFAR-10 and CIFAR-100 [Krizhevsky et al., 2009] contain 60K images each with 10 and 100 distinct classes respectively. We used the default train-test split of 50K-10K. To perform a hyperparameters search, we allocated 5K examples from the training set. To report the results in Section 5.3 in the main text, we use all of the training data (i.e., training set and validation set). In all experiments on these datasets we used Wide Residual Networks [Zagoruyko and Komodakis, 2016] with a widen factor $k = 5$ so it will fit in the GPU. We used the features obtained in the last layer, after applying average pooling, as the input to the GP layer for DKL, DUE, and GDKL. As for the DLVKL baselines, we used an additional linear layer of size 100 which we split into two halves for the mean and variance vectors of the Gaussian. For the NNGP model we used the same network, but without the average pooling layer as it imposed a large computational burden. We note that this step may harm the performance of the NNGP model [Novak et al., 2019]. On these datasets, for all DKL-based methods, we used a dropout rate of 0.3.

Accepted for the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023).

On CIFAR-10 experiments in Section 5.2 we leverage the Dirichlet likelihood function suggested in [Milios et al., 2018] with $\alpha_\epsilon = 0.01$. To make predictions with this likelihood one needs to sample from the posterior of f_*. Hence during test time, we sampled 1024 values. During the training of GDKL, we sampled 256 values as it uses the predictive distribution to train the model. We train all methods for a total of 7000 gradient steps. For GDKL, we use the first 1000 iteration to pre-train the NNGP model hyper-parameters and then train the DKL model for another 6000 iterations. We used SGD with momentum of 0.9, and an initial learning rate of $1e-2$ that drops by a factor of 10 after 60% and 80% of the training (not including the pre-training stage for GDKL). We used a weight decay of $5e-4$ in all DKL-based methods except for GDKL which was set to 0. We did a grid search to select the best hyper-parameters for each method based on the validation set. The DLVKL objective has two KL divergences. We applied a grid search over their coefficients in $\{0.01, 0.1, 1.0\}$. Since DLVKL is based on variational inference, we set the number of inducing points to be the minimum value between the number of examples and 200. The inducing locations were initialized using k-means. As in the official code of this baseline, we used a prior over the latent variable z having a unit variance and a mean value that corresponds to the PCA projection of the input data. For the DUE baseline, we searched over the normalization coefficient and number of power iterations in $\{1, 3\}$, and for the NNGP model we searched over the weight variance in $\{1, 3, 5\}$ while keeping the variance of the bias fixed at 0.2. We found that performance was similar for all values and picked the value 5. For GDKL we also searched over the parameter $\beta \in \{0.1, 1.\}$ and we found that using $\beta = 1.$ generated better results. Note that since this is a multi-output learning setup the KL divergence in GDKL objective results in a summation over the classes. To make it invariant to the number of classes we take an average instead of a sum. This is effectively the same as scaling the KL divergence term by another factor that equals to 0.1. We did not perform data augmentations in these experiments.

In the experiments of Section 5.3, for the most part, we followed the protocol suggested in [van Amersfoort et al., 2021]. Here, we used the Softmax likelihood function for training the deep kernels of all methods. To train the models we used 16 samples from the latent GP when computing the likelihood, and on novel test points, we used 320 samples. On CIFAR-10 we set the number of inducing points to 10, and on CIFAR-100 to 200 for all methods. All inducing locations were initialized using k-means. We set the number of training epochs to 200 with a batch size of 256. To train GDKL we initially sampled 5% of the data to train the hyper-parameters of the NNGP model for 1000 iterations which correspond to ~ 5 epochs with a batch size of 256. Then we train the DKL model for an additional 194 epochs. We used SGD with momentum of 0.9, and an initial learning rate of $1e-1$ that drops by a factor of 10 after 50% and 75% of the training (not including the pre-training stage for GDKL). We used a weight decay of $5e-4$ for all methods. Here we used $\beta = 0.1$ for GDKL. Also, as in the exact setting of GDKL, we approximate the objective in Eq. 10 with MC samples. Note, however, that unlike Eq. 7 where $\mathcal{D}_1$ appears in all terms of the objective, here the corresponding element $\mathcal{B}_1$ appears only in the posterior distribution of the NNGP model. Thus, to be more data efficient, we use two samples. One with $\mathcal{B}_1$ as the "observed data" and another one with $\mathcal{B}_2$ as the "observed data". In DUE and DKL, we also searched over the coefficient of the KL divergence in the variational ELBO objective in $\{0.1, 1.0\}$. We used random cropping and random horizontal flip for data augmentation.

B THE GDKL OBJECTIVE

In Section 3.1 we presented to following objective:

$$\mathbb{E}_{\mathbf{x}_*, y_* \sim \mathcal{D}_2} D_{KL}[q_\theta(f_*|\mathbf{x}_*, \mathcal{D}_1)||p(f_*|y_*, \mathbf{x}_*, \mathcal{D}_1)]. \tag{1}$$

We now show that it is equivalent to the objective of Eq. 6 in the main text.

$$\begin{aligned}
&\mathbb{E}_{\mathbf{x}_*, y_* \sim \mathcal{D}_2} D_{KL}[q_\theta(f_*|\mathbf{x}_*, \mathcal{D}_1)||p(f_*|\mathbf{x}_*, y_*, \mathcal{D}_1)] \\
&= \mathbb{E}_{\mathbf{x}_*, y_* \sim \mathcal{D}_2} \mathbb{E}_{q_\theta(f_*|\mathbf{x}_*, \mathcal{D}_1)}[\log \ \frac{q_\theta(f_*|\mathbf{x}_*, \mathcal{D}_1)}{p(f_*|\mathbf{x}_*, y_*, \mathcal{D}_1)}] \\
&= \mathbb{E}_{\mathbf{x}_*, y_* \sim \mathcal{D}_2} \mathbb{E}_{q_\theta(f_*|\mathbf{x}_*, \mathcal{D}_1)}[\log \ q_\theta(f_*|\mathbf{x}_*, \mathcal{D}_1) - \log \ \frac{p(y_*|f_*)p(f_*|\mathbf{x}_*, \mathcal{D}_1)}{p(\mathbf{x}_*, \mathcal{D}_1)}] \\
&= \mathbb{E}_{\mathbf{x}_*, y_* \sim \mathcal{D}_2} \mathbb{E}_{q_\theta(f_*|\mathbf{x}_*, \mathcal{D}_1)}[\log \ q_\theta(f_*|\mathbf{x}_*, \mathcal{D}_1) - \log \ p(y_*|f_*) - \log \ p(f_*|\mathbf{x}_*, \mathcal{D}_1) + \log \ p(\mathbf{x}_*, \mathcal{D}_1)] \\
&\propto \mathbb{E}_{\mathbf{x}_*, y_* \sim \mathcal{D}_2} \mathbb{E}_{q_\theta(f_*|\mathbf{x}_*, \mathcal{D}_1)}[-\log \ p(y_*|f_*)] + D_{KL}[q_\theta(f_*|\mathbf{x}_*, \mathcal{D}_1)||p(f_*|\mathbf{x}_*, \mathcal{D}_1)].
\end{aligned} \tag{2}$$

Where, in the third step we used Bayes rule, and in the last step we dropped the constant factor $\log \ p(\mathbf{x}_*, \mathcal{D}_1)$ which doesn't effect the optimization process.

For a Gaussian likelihood, the posterior predictive distributions can be derived using standard Gaussian algebra

[Rasmussen and Williams, 2006]. For instance,

$$
\begin{aligned}
&p(f_*|\mathbf{x}_*, \mathcal{D}_1) = \mathcal{N}(\mu_*^p, (\sigma_*^p)^2), \\
&\mu_*^p = \mathbf{k}_*^T(\mathbf{K} + \sigma_n^2 \boldsymbol{I})^{-1}\mathbf{y}, \\
&(\sigma_*^p)^2 = k_{**} - \mathbf{k}_*^T(\mathbf{K} + \sigma_n^2 \boldsymbol{I})^{-1}\mathbf{k}_*.
\end{aligned} \tag{3}
$$

Where, $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, $k_{**} = k(\mathbf{x}_*, \mathbf{x}_*)$, and $\mathbf{k}_*[i] = k(\mathbf{x}_i, \mathbf{x}_*)$. In a similar fashion $q_\theta(f_*|\mathbf{x}_*, \mathcal{D}_1) = \mathcal{N}(f_*|\mu_*^q, (\sigma_*^q)^2)$ can be obtained.

Now, the D_{KL} term has the following closed-form solution:

$$
D_{KL}[q_\theta(f_*|\mathbf{x}_*, \mathcal{D}_1)||p(f_*|\mathbf{x}_*, \mathcal{D}_1)] = \log\ \frac{\sigma_*^p}{\sigma_*^q} + \frac{(\sigma_*^q)^2 + (\mu_*^q - \mu_*^p)^2}{2(\sigma_*^p)^2} - \frac{1}{2}. \tag{4}
$$

Similarly, the expected log-likelihood term can be computed analytically using the following:

$$
\mathbb{E}_{q_\theta(f_*|\mathbf{x}_*, \mathcal{D}_1)}[-\log\ p(y_*|f_*)] = \frac{1}{2}(\log\ 2\pi + \log\ \sigma_n^2 + \frac{(y_* - \mu_*^q)^2 + (\sigma_*^q)^2}{\sigma_n^2}). \tag{5}
$$

C COMPUTATIONAL CONSIDIRATIONS

In this section we address the computational complexity of GDKL from two aspects: (1) the scaling limitations imposed by the NNGP kernel, and (2) comparison to baseline methods.

Scaling limitations imposed by the NNGP kernel. GDKL leverages NNGP kernels to learn the model. One may wonder if that may pose a limit to GDKL as computing the NNGP kernel can be costly. To address this concern we provide two important computational aspects to showcase that it is not an issue for GDKL. Furthermore, we argue that GDKL posses computational advantages over using NNGPs directly, aside from the added benefit in performance.

- First, as with standard NNGPs, GDKL inherits the flexibility in choosing the architecture of the NNGP and other design choices, such as the data resolution on which this kernel is computed. However, unlike traditional NNGPs where the posterior is heavily influenced by the NNGP kernel, in GDKL one may choose simpler kernels that possibly can be computed more efficiently without it resulting in a significant performance drop. This is because the NNGP basically serves as a prior in our model, aimed at calibrating the uncertainties of the DKL model. Hence, as we see it, GDKL can provide practitioners with the freedom to choose an NNGP model based on their hardware constraints, without compromising on model performance significantly. To validate that point, in Table 1 we present the effect of using only one residual block in each group, instead of four, in the kernel obtained by the wide-residual network architecture which we used throughout. The table shows the results in terms of test accuracy under the setup of Section 5.2, but a similar trend was observed in terms of log-likelihood as well. The results in the table indicate a clear advantage, albeit small but still statistically significant, to the deeper NNGP model compared to the shallower one. However, when using the shallower NNGP model in the training process of GDKL, it has almost no effect on the test accuracy of GDKL.

Table 1: Effect of infinite network depth - test accuracy on CIFAR-10 with $\{50, 100, 200, 400, 800\}$ training examples.

	50	100	200	400	800
NNGP - One Res. Block	18.80 ± 0.11	21.10 ± 0.11	27.71 ± 0.13	32.18 ± 0.11	36.34 ± 0.09
NNGP - Four Res. Blocks	18.87 ± 0.16	22.47 ± 0.16	28.94 ± 0.13	34.39 ± 0.09	38.73 ± 0.07
GDKL - One Res. Block	19.41 ± 0.05	26.70 ± 1.11	36.52 ± 0.94	42.60 ± 0.53	49.34 ± 0.52
GDKL - Four Res. Blocks	19.35 ± 0.65	26.53 ± 0.98	36.45 ± 0.72	42.97 ± 0.71	49.75 ± 0.94

- Second, in terms of computation of the kernel. When dealing with small to medium-sized datasets, computing the NNGP kernel is not typically expensive and can be done offline before training. However, for larger datasets, scalability becomes more challenging. In these cases, usually one will need to deal with scalability issues of GPs in general, and the common practice is to use inducing point (IP) methods such as the one we proposed in the paper. In the IP variant of GDKL, batches are sampled during training and the NNGP kernel depends only on the examples in them. Therefore, one option is to compute the NNGP kernel online based on the examples in each batch. Since the batch

size is usually small (e.g., 256), efficient optimization packages (e.g., [Novak et al., 2020]) can be utilized to compute these kernel matrices in a relatively efficient manner. However, this approach may still be slower than training standard DKLs. Therefore, a further improvement can be done with proper engineering work. One can pre-compute the kernel of the examples in each batch offline before training by taking into account the stochasticity in forming batches during training. These pre-calculated kernels can then be used during training of GDKL. This is an advantage of GDKL over the standard NNGP model, which requires computation of the full kernel matrix.

Finally, we would like to highlight two important points. First, although the training of GDKL can be slower compared to DKL, when making predictions the models are equivalent. This is unlike the NNGP model which scales linearly with the number of training points. Second, there are ongoing efforts to scale NNGP models to larger datasets, as evident by recent studies such as [Adlam et al., 2023]. These advancements in scaling NNGP models could potentially be leveraged in GDKL as well, if needed, to further improve its scalability and applicability to larger datasets.

Comparison to baseline methods. The GDKL objective consists of two components in the loss function: a predictive distribution term and a KL divergence term. The most computationally intensive factor in calculating both terms is the inversion of the DKL and the NNGP kernel matrices, each with a complexity of $\mathcal{O}((n/2)^3)$ in the exact case. The division by 2 is due to GDKL partitioning the data into two halves at each iteration, using one half for making predictions on the other half. When employing m inducing points, the complexity of the computation involves the inverse of the kernel over the inducing locations and the inverse of the NNGP kernel over half of the examples in the batch, resulting in a complexity of $\mathcal{O}(m^3 + (\mathcal{B}/2)^3)$, where $\mathcal{B}$ is the batch size. A potential speedup can be achieved by pre-calculating the inverse of the NNGP kernel offline before training, and using it during training. This way, during training the model's computational speed would be similar to standard DKL models.

To estimate the training time difference between the methods we measured the average time per-iteration in seconds on CIFAR-10 based on 100 iterations five times. In Table 2 we present these timing along with the constant time taken for computing the NNGP kernel before training. According to the data presented in the table, GDKL tends to exhibit slightly slower performance compared to DKL and DUE. However, it's worth noting that the current implementation of the code is not highly optimized, and there are several aspects that can be improved, such as the computation time for the NNGP kernel and the average iteration time.

Table 2: Average run time (Sec.) on CIFAR-10 with $\{50, 100, 200, 400, 800\}$ training examples. Results are based on 100 iterations done 5 times.

	50	100	200	400	800
NNGP (One Time)	5.44 ± 0.32	5.51 ± 0.07	5.91 ± 0.10	8.12 ± 0.23	16.70 ± 0.16
DKL	0.06 ± 0.00	0.12 ± 0.00	0.22 ± 0.00	0.44 ± 0.00	0.65 ± 0.00
DLVKL	0.10 ± 0.01	0.18 ± 0.00	0.29 ± 0.00	0.56 ± 0.00	0.84 ± 0.00
DUE	0.08 ± 0.00	0.14 ± 0.00	0.23 ± 0.00	0.45 ± 0.00	0.67 ± 0.00
GDKL	0.07 ± 0.00	0.14 ± 0.00	0.24 ± 0.00	0.48 ± 0.00	0.76 ± 0.04

D ADDITIONAL EXPERIMENTS

D.1 OBJECTIVE FUNCTIONS ANALYSIS

Table 3: Ablation on objective functions - test results on the UCI datasets based on ten random splits.

	Boston		Energy		Concrete	
	LL (↑)	RMSE (↓)	LL (↑)	RMSE (↓)	LL (↑)	RMSE (↓)
NNGP	-2.49 ± 0.16	3.19 ± 0.70	-1.07 ± 0.04	0.69 ± 0.02	-3.15 ± 0.04	4.71 ± 0.49
ℓ_{dist}	**-2.51 ± 0.13**	3.32 ± 0.52	-1.23 ± 0.04	0.72 ± 0.08	-3.03 ± 0.14	5.02 ± 0.58
ℓ_{pred}	-499. ± 229.	3.27 ± 0.92	-1.78 ± 1.10	**0.28 ± 0.07**	-6.72 ± 1.52	**4.21 ± 0.90**
GDKL	**-2.49 ± 0.20**	**3.03 ± 0.54**	**-0.50 ± 0.07**	**0.32 ± 0.06**	**-2.93 ± 0.10**	4.58 ± 0.57

Here we study the effect of using the GDKL objective vs ℓ_{dist} and ℓ_{pred} which were presented in Section 3.1. We evaluated all methods on the UCI datasets Boston, Concrete, and Energy according to the setup described in Section 5.1. The results

are presented in Table 3. The table shows that the results are in agreement with our intuition. First, ℓ_{dist} behavior is similar to that of the NNGP, the model that it tries to distill. Second, ℓ_{pred} clearly overfits as indicated by the log-likelihood values, yet according to the RMSE it is able to maintain a good mean prediction. And lastly, our proposed approach balances well between these two edges, it presents the best results on both metrics in almost all cases.

D.2 COMPARISON TO A STANDARD NEURAL NETWORK

Table 4: Comparison to a standard NN - test results on CIFAR-10 with $\{50, 100, 200, 400, 800\}$ training examples.

	Log-Likelihood					Accuracy				
	50	100	200	400	800	50	100	200	400	800
NN	-4.55 ± 0.17	-4.63 ± 0.27	-4.20 ± 0.26	-3.76 ± 0.34	-3.28 ± 0.21	19.44 ± 1.17	20.94 ± 1.90	27.82 ± 1.67	34.96 ± 2.48	42.81 ± 2.06
GDKL (Ours)	**-2.29 ± 0.01**	**-2.08 ± 0.03**	**-1.83 ± 0.01**	**-1.66 ± 0.01**	**-1.49 ± 0.01**	19.35 ± 0.65	**26.53 ± 0.98**	**36.45 ± 0.72**	**42.97 ± 0.71**	**49.75 ± 0.94**

Here we compare GDKL to a standard NN on the CIFAR-10 dataset under the setup outlined in Section 5.2. We present the test results when varying the number of training examples from 50 to 800 based on ten random seeds in Table 4. From the table, GDKL demonstrates superior performance compared to a standard NN in terms of both log-likelihood and accuracy in almost all cases. Furthermore, when cross referencing these results with those in Figure 3, in general, GP-based methods outperform standard NNs in these experiments conducted under low-data regime conditions.

D.3 RELIABILITY DIAGRAMS

Here we quantify the confidence through calibration for GDKL and baseline methods on the CIFAR-10 dataset in the setting described in Section 5.2. We use reliability diagrams and the following metrics [Brier, 1950, Guo et al., 2017]: (1) Expected Calibration Error (ECE), which measures the weighted average distance between the classifier confidence and accuracy; (2) Maximum Calibration Error (MCE) which measures the maximum distance between the classifier confidence and accuracy; and (3) Brier score (BRI) which measures the average squared error between the prediction probabilities and the actual labels. Figure 1 shows that GDKL is best calibrated across all metrics in all cases when $n \geq 200$, and on smaller dataset sizes only the NNGP model is better. We note that temperature scaling can improve calibration, yet finding the right temperature requires having an additional validation set.

D.4 FULL RESULTS

In this section, we provide full numerical results for the experiments described in Sections 5.1 and 5.2.

Table 5: Train results on small UCI datasets based on ten random splits.

	Boston		Energy		Concrete	
	LL (↑)	RMSE (↓)	LL (↑)	RMSE (↓)	LL (↑)	RMSE (↓)
DKL	1.45 ± 0.01	0.00 ± 0.00	1.36 ± 0.00	0.00 ± 0.00	-1.89 ± 0.47	1.28 ± 0.03
NNGP	-2.07 ± 0.05	1.66 ± 0.09	1.00 ± 0.01	0.01 ± 0.00	-2.96 ± 0.01	2.56 ± 0.07
GP-RBF	-1.85 ± 0.09	1.31 ± 0.16	-0.18 ± 0.03	0.27 ± 0.01	-2.15 ± 0.05	1.70 ± 0.01
GDKL	-1.91 ± 0.04	1.04 ± 0.06	-0.08 ± 0.04	0.03 ± 0.00	-2.64 ± 0.02	2.71 ± 0.07

Table 6: Test results on small UCI datasets based on ten random splits.

	Boston		Energy		Concrete	
	LL (↑)	RMSE (↓)	LL (↑)	RMSE (↓)	LL (↑)	RMSE (↓)
DKL	-553. ± 265.	3.12 ± 0.71	-3.59 ± 2.09	0.32 ± 0.07	-3.89 ± 0.66	3.96 ± 0.63
NNGP	-2.49 ± 0.16	3.19 ± 0.70	-1.07 ± 0.04	0.69 ± 0.02	-3.15 ± 0.04	4.71 ± 0.49
GP-RBF	-2.39 ± 0.21	2.82 ± 0.64	-0.51 ± 0.13	0.40 ± 0.05	-2.90 ± 0.23	5.59 ± 0.80
GDKL	-2.47 ± 0.14	3.03 ± 0.49	-0.31 ± 0.06	0.28 ± 0.03	-2.92 ± 0.09	4.58 ± 0.60

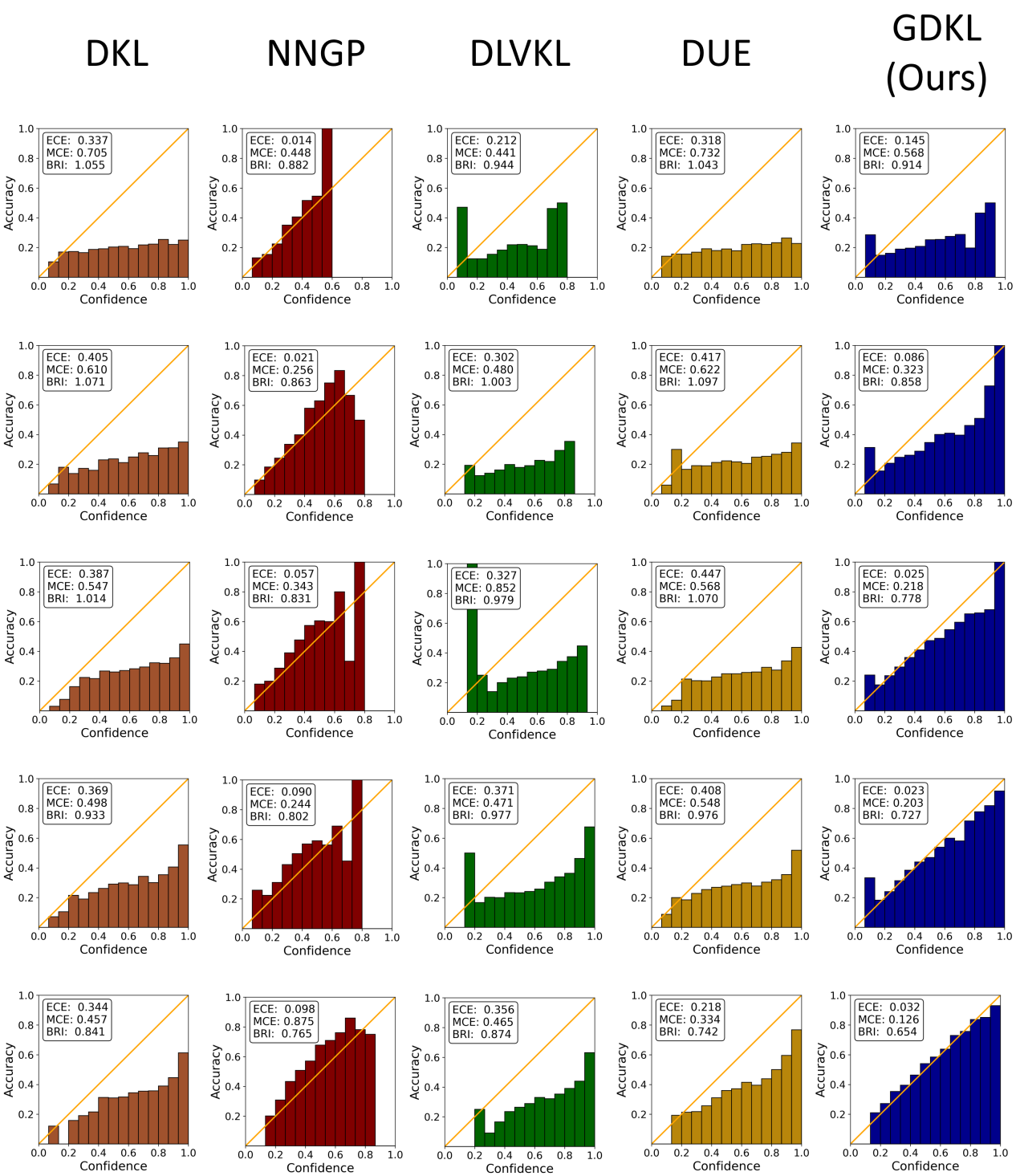

Figure 1: Reliability diagrams on CIFAR-10 test set for the experiments in Section 5.2 with training examples ranging from 50 (top row) to 800 (bottom row) examples.

Table 7: Test results on Buzz, CTSlice, and CIFAR-10 - 50 examples.

	Buzz		CTSlice		CIFAR-10	
	LL (↑)	RMSE (↓)	LL (↑)	RMSE (↓)	LL (↑)	Acc. (↑)
DKL	-421. ± 208.	1.61 ± 0.26	-283. ± 577.	20.07 ± 3.54	- 2.61 ± 0.05	20.26 ± 0.97
NNGP	-1.29 ± 0.04	1.06 ± 0.13	-3.94 ± 0.05	12.35 ± 0.93	- 2.23 ± 0.00	18.87 ± 0.16
GP-RBF	-1.61 ± 0.27	1.30 ± 0.16	-4.53 ± 0.00	22.34 ± 0.15	– ± –	– ± –
DLVKL	– ± –	– ± –	– ± –	– ± –	- 2.42 ± 0.04	18.79 ± 0.86
DUE	– ± –	– ± –	– ± –	– ± –	- 2.64 ± 0.07	20.18 ± 0.85
GDKL	-1.23 ± 0.09	0.86 ± 0.08	-3.92 ± 0.11	11.73 ± 1.10	- 2.29 ± 0.02	19.35 ± 0.65

Table 8: Test results on Buzz, CTSlice, and CIFAR-10 - 100 examples.

	Buzz		CTSlice		CIFAR-10	
	LL (↑)	RMSE (↓)	LL (↑)	RMSE (↓)	LL (↑)	Acc. (↑)
DKL	-336. ± 185.	1.38 ± 0.25	-518. ± 350.	13.10 ± 5.09	- 2.73 ± 0.04	24.67 ± 1.26
NNGP	-1.18 ± 0.04	0.98 ± 0.14	-3.75 ± 0.03	10.25 ± 0.46	- 2.14 ± 0.00	22.47 ± 0.17
GP-RBF	-1.36 ± 0.27	1.10 ± 0.09	-4.52 ± 0.00	22.33 ± 0.14	– ± –	– ± –
DLVKL	– ± –	– ± –	– ± –	– ± –	- 2.40 ± 0.07	22.81 ± 1.68
DUE	– ± –	– ± –	– ± –	– ± –	- 2.77 ± 0.14	24.32 ± 1.53
GDKL	-1.14 ± 0.05	0.77 ± 0.06	-3.64 ± 0.08	9.05 ± 0.77	- 2.08 ± 0.03	26.53 ± 0.98

Table 9: Test results on Buzz, CTSlice, and CIFAR-10 - 200 examples.

	Buzz		CTSlice		CIFAR-10	
	LL (↑)	RMSE (↓)	LL (↑)	RMSE (↓)	LL (↑)	Acc. (↑)
DKL	-178. ± 127.	1.39 ± 0.21	-459. ± 210.	8.18 ± 0.74	- 2.58 ± 0.04	33.65 ± 1.24
NNGP	-1.09 ± 0.02	0.91 ± 0.10	-3.58 ± 0.05	8.89 ± 0.36	- 2.02 ± 0.00	28.94 ± 0.13
GP-RBF	-1.09 ± 0.08	0.90 ± 0.11	-4.42 ± 0.19	20.21 ± 3.41	– ± –	– ± –
DLVKL	– ± –	– ± –	– ± –	– ± –	- 2.42 ± 0.09	27.59 ± 1.68
DUE	– ± –	– ± –	– ± –	– ± –	- 2.56 ± 0.21	33.10 ± 1.18
GDKL	-1.08 ± 0.04	0.77 ± 0.06	-3.45 ± 0.07	7.73 ± 0.66	- 1.83 ± 0.01	36.45 ± 0.72

Table 10: Test results on Buzz, CTSlice, and CIFAR-10 - 400 examples.

	Buzz		CTSlice		CIFAR-10	
	LL (↑)	RMSE (↓)	LL (↑)	RMSE (↓)	LL (↑)	Acc. (↑)
DKL	-136. ± 82.2	1.34 ± 0.35	-285. ± 130.	7.22 ± 2.23	- 2.36 ± 0.10	40.96 ± 0.89
NNGP	-1.00 ± 0.01	0.86 ± 0.09	-3.39 ± 0.03	7.57 ± 0.24	- 1.92 ± 0.00	34.39 ± 0.09
GP-RBF	-0.97 ± 0.03	0.79 ± 0.05	-3.77 ± 0.29	10.84 ± 0.86	– ± –	– ± –
DLVKL	– ± –	– ± –	– ± –	– ± –	- 2.43 ± 0.09	33.48 ± 1.54
DUE	– ± –	– ± –	– ± –	– ± –	- 2.20 ± 0.21	40.33 ± 0.76
GDKL	-1.01 ± 0.03	0.71 ± 0.04	-3.20 ± 0.05	5.98 ± 0.42	- 1.66 ± 0.01	42.97 ± 0.71

Table 11: Test results on Buzz, CTSlice, and CIFAR-10 - 800 examples.

	Buzz		CTSlice		CIFAR-10	
	LL (↑)	RMSE (↓)	LL (↑)	RMSE (↓)	LL (↑)	Acc. (↑)
DKL	-114. ± 94.1	1.09 ± 0.12	-246. ± 110.	5.70 ± 1.47	- 2.13 ± 0.10	48.57 ± 0.59
NNGP	-0.94 ± 0.01	0.81 ± 0.13	-3.15 ± 0.02	6.25 ± 0.19	- 1.80 ± 0.06	38.73 ± 0.07
GP-RBF	-0.89 ± 0.01	0.72 ± 0.01	-3.14 ± 0.11	7.61 ± 0.56	– ± –	– ± –
DLVKL	– ± –	– ± –	– ± –	– ± –	- 2.22 ± 0.06	43.52 ± 0.75
DUE	– ± –	– ± –	– ± –	– ± –	- 1.91 ± 0.08	49.22 ± 0.79
GDKL	-0.95 ± 0.02	0.67 ± 0.03	-2.99 ± 0.06	4.87 ± 0.45	- 1.49 ± 0.02	49.75 ± 0.94

References

Ben Adlam, Jaehoon Lee, Shreyas Padhy, Zachary Nado, and Jasper Snoek. Kernel regression with infinite-width neural networks on millions of examples. *arXiv preprint arXiv:2303.05420*, 2023.

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

Jacob Gardner, Geoff Pleiss, Kilian Q Weinberger, David Bindel, and Andrew G Wilson. GPyTorch: Blackbox matrix-matrix Gaussian process inference with gpu acceleration. *Advances in neural information processing systems*, 31, 2018.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330. PMLR, 2017.

Diederik P. Kingma and Jimmy Ba. ADAM: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR*, 2015.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Florent Leclercq. Bayesian optimization for likelihood-free cosmological inference. *Physical Review D*, 98(6):063511, 2018.

Dimitrios Milios, Raffaello Camoriano, Pietro Michiardi, Lorenzo Rosasco, and Maurizio Filippone. Dirichlet-based Gaussian processes for large-scale calibrated classification. *Advances in Neural Information Processing Systems*, 31, 2018.

Roman Novak, Lechao Xiao, Yasaman Bahri, Jaehoon Lee, Greg Yang, Jiri Hron, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Bayesian deep convolutional networks with many channels are Gaussian processes. In *7th International Conference on Learning Representations, ICLR*, 2019.

Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A. Alemi, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. Neural tangents: Fast and easy infinite neural networks in python. In *8th International Conference on Learning Representations, ICLR*, 2020.

Sebastian W Ober, Carl E Rasmussen, and Mark van der Wilk. The promises and pitfalls of deep kernel learning. In *Uncertainty in Artificial Intelligence*, pages 1206–1216. PMLR, 2021.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

Joost van Amersfoort, Lewis Smith, Andrew Jesson, Oscar Key, and Yarin Gal. On feature collapse and deep kernel learning for single forward pass uncertainty. *arXiv preprint arXiv:2102.11409*, 2021.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference BMVC*, 2016.