APPENDIX C EXPERIMENT DETAILS

C.1 GENERAL SETUP

For all datasets, we normalize input values to be between [0, 1] and standardize output values to have mean 0 and variance 1. We also use Monte-Carlo based Expected Improvement as our acquisition function.

GP: For single-objective problems, we use GPs with a Matérn $5/2$ kernel, adaptive scale, a length-scale prior of $\text{Gamma}(3, 6)$, and an output-scale prior of $\text{Gamma}(2.0, 0.15)$, which combine with the marginal likelihood to form a posterior which we optimize for hyperparameter learning. For multi-objective problems, we use the same GP to independently model each objective.

I-BNN: We use I-BNNs with 3 hidden layers and the ReLU activation function. We set the variance of the weights to 10.0, and the variance of the bias to 1.6. For multi-objective problems, we independently model each objective.

For each of the following surrogate models which include neural networks, we use MLPs with 3 hidden layers and 128 nodes each. During each iteration of Bayesian optimization, we use a grid search over prior variance (0.1, 1.0, 10.0) and likelihood variance (0.1, 0.32, 1.0) to find the combination which maximizes $\mathcal{L}$, where $\mathcal{L}$ represents the likelihood of the surrogate model on a random 20% of the existing function evaluations when the model is trained on the other 80%.

DKL: We set up the base kernel using the same Matérn $5/2$ kernel that we use for GPs. For the feature extractor, we use the model parameters as explained above. For multi-objective problems, we independently model each objective.

HMC: We use HMC with an adaptive step size, and we choose the architecture as explained above. We model multi-objective problems by setting the number of nodes in the final layer equal to the number of objectives.

SGHMC: We use SGHMC with minibatch size of 5 and neural network architecture as indicated above. We follow the implementation in Springenberg et al. (2016) and use scale-adaptive SGHMC with a heteroskedastic likelihood variance as determined by the output of the neural network. We model multi-objective problems by setting the number of nodes in the final layer equal to the number of objectives.

LLA: We use the model architecture as explained. We model multi-objective problems by setting the number of nodes in the final layer equal to the number of objectives.

ENSEMBLE: We use an ensemble of 5 models, each with the architecture explained above. Each model is trained on a random 80% of the function evaluations. We model multi-objective problems by setting the number of nodes in the final layer equal to the number of objectives.

We run multiple trials for all experiments, where each trial starts with a different set of initial function evaluations drawn using a Sobol sampler.

C.2 SYNTHETIC BENCHMARKS

Branin We ran 10 trials using batch size 5 with 10 initial points.

Hartmann We ran 10 trials using batch size 10 with 10 initial points.

Ackley We ran 10 trials using batch size 10 with 10 initial points.

BraninCurrin We ran 10 trials using batch size 10 with 10 initial points.

DTLZ1 We ran 10 trials using batch size 5 with 10 initial points.

DTLZ5 We ran 10 trials using batch size 1 with 10 initial points.

C.3 REAL-WORLD BENCHMARKS

PDE Optimization We ran 10 trials using batch size 1 with 5 initial points.

Interferometer We ran 10 trials using batch size 10 with 10 initial points.

Lunar Lander We ran 10 trials using batch size 50 with 50 initial points.

Pest Control We ran 10 trials using batch size 4 with 20 initial points.

Cell Coverage We ran 10 trials using batch size 5 with 10 initial points.

Oil Spill Sorbent We ran 10 trials using batch size 10 with 10 initial points.

C.4 High-Dimensional Experiments

Polynomial We ran 10 trials using batch size 10 with 100 initial points.

Neural Network Draw We ran 10 trials using batch size 10 with 100 initial points.

Knowledge Distillation We ran 10 trials using batch size 10 with 100 initial points.

C.5 Neural Architecture Search

We used SMAC (Lindauer et al., 2022) to find the optimal hyperparameters of HMC for Cell Coverage, Pest Control, and DTLZ5 benchmark problems using Bayesian optimization. For each benchmark, our new objective function was the average maximum value found for three runs of Bayesian optimization using the same problem setup as detailed above. We used the hyperparameter optimization facade in SMAC, and for each problem, we used Bayesian optimization to select 200 different HMC architectures to find the optimal combination from the following set of possible values:

- Network width: [32, 512] (continuous)
- Network depth: $\{1, 2, 3, 4, 5, 6\}$ (discrete)
- Network activation: {"ReLU", "tanh"} (discrete)
- $\log_{10}$ of likelihood variance: [-3.0, 2.0] (continuous)
- $\log_{10}$ of prior variance: [-3.0, 2.0] (continuous)

We then use the optimal architecture and run the benchmark for 10 trials with the same setup as described in Appendix C.2 and Appendix C.3 to compare the results of HMC with and without neural architecture search.

For cell coverage, the final HMC model was an MLP with 5 hidden layers, 184 parameters per layer, tanh activation, likelihood variance of 26.3, and prior variance of 0.54.

For pest control, the final HMC model was an MLP with 1 hidden layer of size 321, tanh activation, likelihood variance of 0.26, and prior variance of 0.31.

For DTLZ5, the final HMC model was an MLP with 3 hidden layers, 297 parameters per layer, relu activation, likelihood variance of 0.01, and prior variance of 0.30.

Appendix D Further Empirical Results

D.1 BNN Architecture

D.1.1 Neural Architecture Search

To further investigate the impact of architecture on the performance of BNNs, we conduct an extensive neural architecture search over a selection of the benchmark problems, varying the width, depth, prior variance, likelihood variance, and activation function. For our experiments, we use SMAC3 (Lindauer et al., 2022), a framework which uses Bayesian optimization to select the best hyperparameters, and we detail the experiment setup in Appendix C. While a thorough search over architectures is often impractical for realistic settings of Bayesian optimization since it requires a very large number of function evaluations, we use this experiment to demonstrate the flexibility of BNNs and to showcase its potential when the design is well-suited for the problem.

We show the effect of neural architecture search on HMC surrogate models in Figure A.1. On the Cell Coverage problem, the architecture search did not drastically change the performance of HMC. In contrast, extensively optimizing the hyperparameters made a significant difference on the Pest Control problem, leading to HMC finding higher rewards than GPs while using fewer function evaluations; however, on this problem, I-BNN, which does not require specifying an architecture, still performs best. Neural architecture search was also able to improve the results on DTLZ5, leading HMC to be competitive with other surrogate models such as I-BNNs and DKL. The difference in the benefits of the search may be attributed to some problems having less inherent structure than others, where extensive hyperparameter optimization may not be as necessary. Additionally, our original HMC surrogate model choice may already have been a suitable choice for some problems, so an extensive search over architectures may not significantly improve the performance.

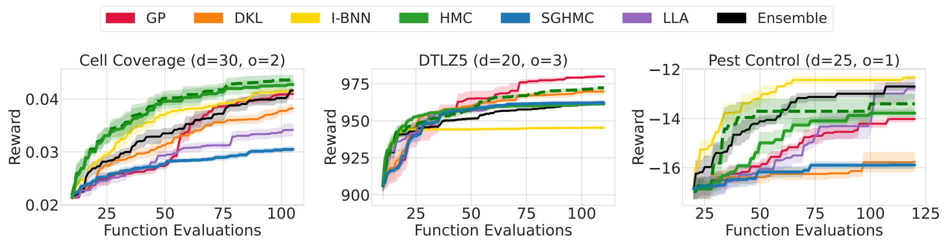
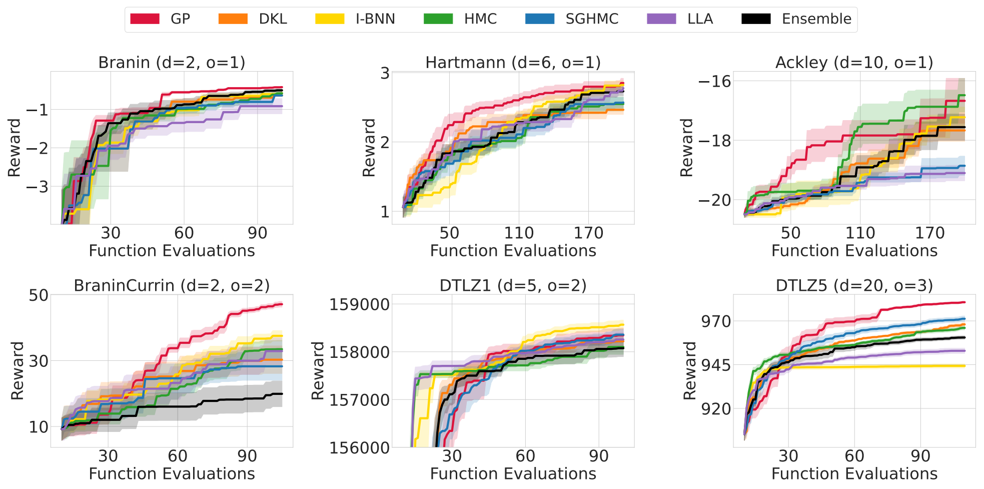


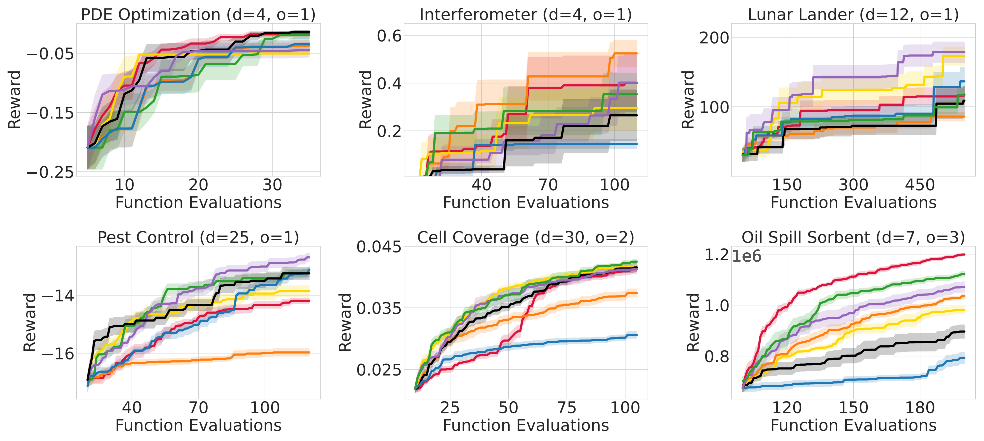
Figure A.1: **The impact of neural architecture search on HMC is problem-dependent.** The dashed green line indicates the performance of HMC after an extensive neural architecture search, compared to the solid green line representing the HMC model selected from a much smaller pool of hyperparameters. We see that it has minimal impact on Cell Coverage (left), moderate impact on DTLZ5 (center), and extensive impact on Pest Control (right), even outperforming GPs. For each benchmark, we include d for the number of input dimensions, and o for the number of objectives. We plot the mean and one standard error of the mean over 10 trials.

D.1.2 Smaller Architecture

Here, we benchmark the performance of BNNs using the relatively small architecture specified by Kristiadi et al. (2023): an MLP with 2 hidden layers of size 50 with ReLU activation. In the main text, our experiments used MLPs with 3 hidden layers of size 128 with tanh activation. Our experiments show that the larger BNN from our main text typically leads to better performance across datasets. However, even with the smaller architecture, we find that the relative performance of the surrogate models mostly remains consistent. We still find that HMC often outperforms other approximate inference methods such as deep ensembles, and we see that SGHMC often finds suboptimal rewards. We find that LLA seems to have slightly improved performance using this smaller architecture, especially over many of the real datasets.



(a) Synthetic Benchmarks



(b) Real-World Benchmarks

Figure A.2: **BNN results with a different architecture**. We show the performance of BNN surrogate models with the relatively small architecture specified by Kristiadi et al. (2023): an MLP with 2 hidden layers of size 50 with ReLU activation. For each benchmark, we include d for the number of input dimensions, and o for the number of objectives. We plot the mean and one standard error of the mean over 10 trials.

D.2 Ablation Studies

To better understand the quality of the mean estimates and uncertainty estimates of different surrogate models, we conducted ablation studies by creating hybrid models which combine the mean estimate of one surrogate with the uncertainty estimate of another. When we compare this hybrid model with each individual model, we are able to get insight into the relative performance of the mean and uncertainty estimates. We provide the results for GP vs HMC, the gold standard inference for BNNs, and GP vs I-BNN, the model with the most success in high-dimensions.

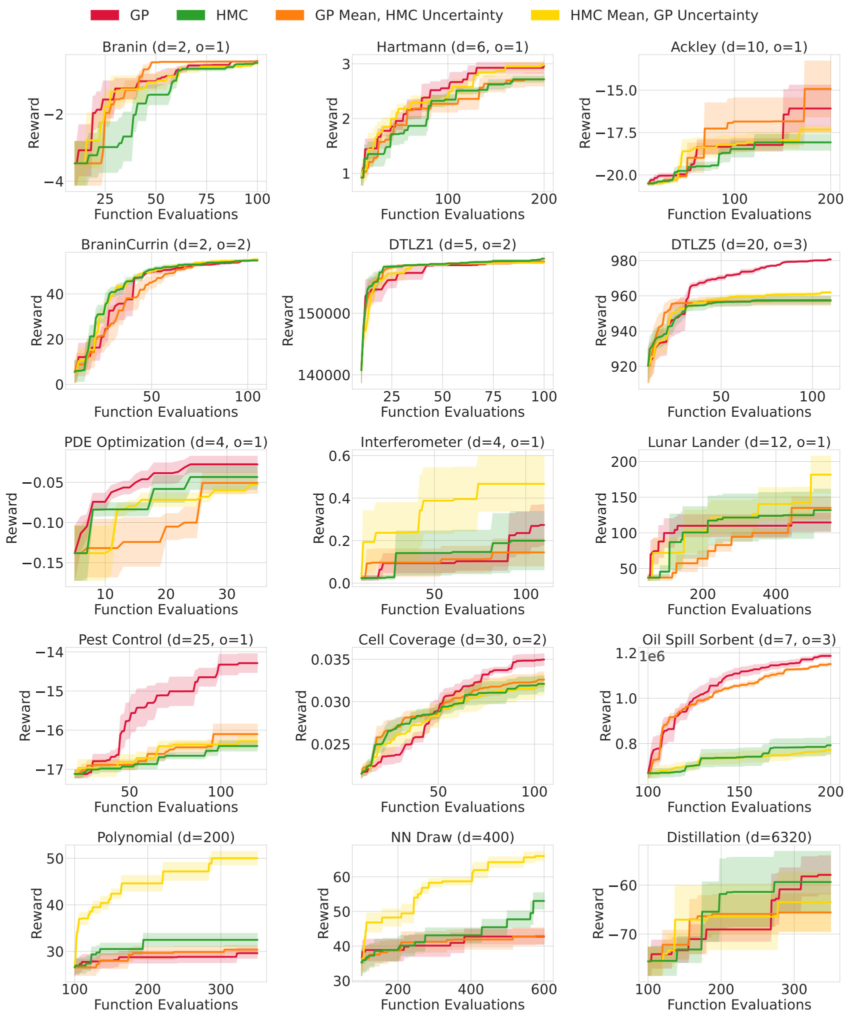


Figure A.3: When we compare GP vs HMC in lower dimensions, we see that mean and uncertainty estimates of the GP need to work together to achieve optimal results, and HMC does not significantly improve when we instead use the GP mean or uncertainty estimates. In contrast, in higher dimensions, the best performing model is the HMC-Mean GP-Uncertainty hybrid, suggesting HMC has better mean estimates while GPs have better uncertainty estimates in this setting. We plot the mean and one standard error of the mean over 10 trials, d refers to the number of input dimensions, and o refers to the number of output dimensions.

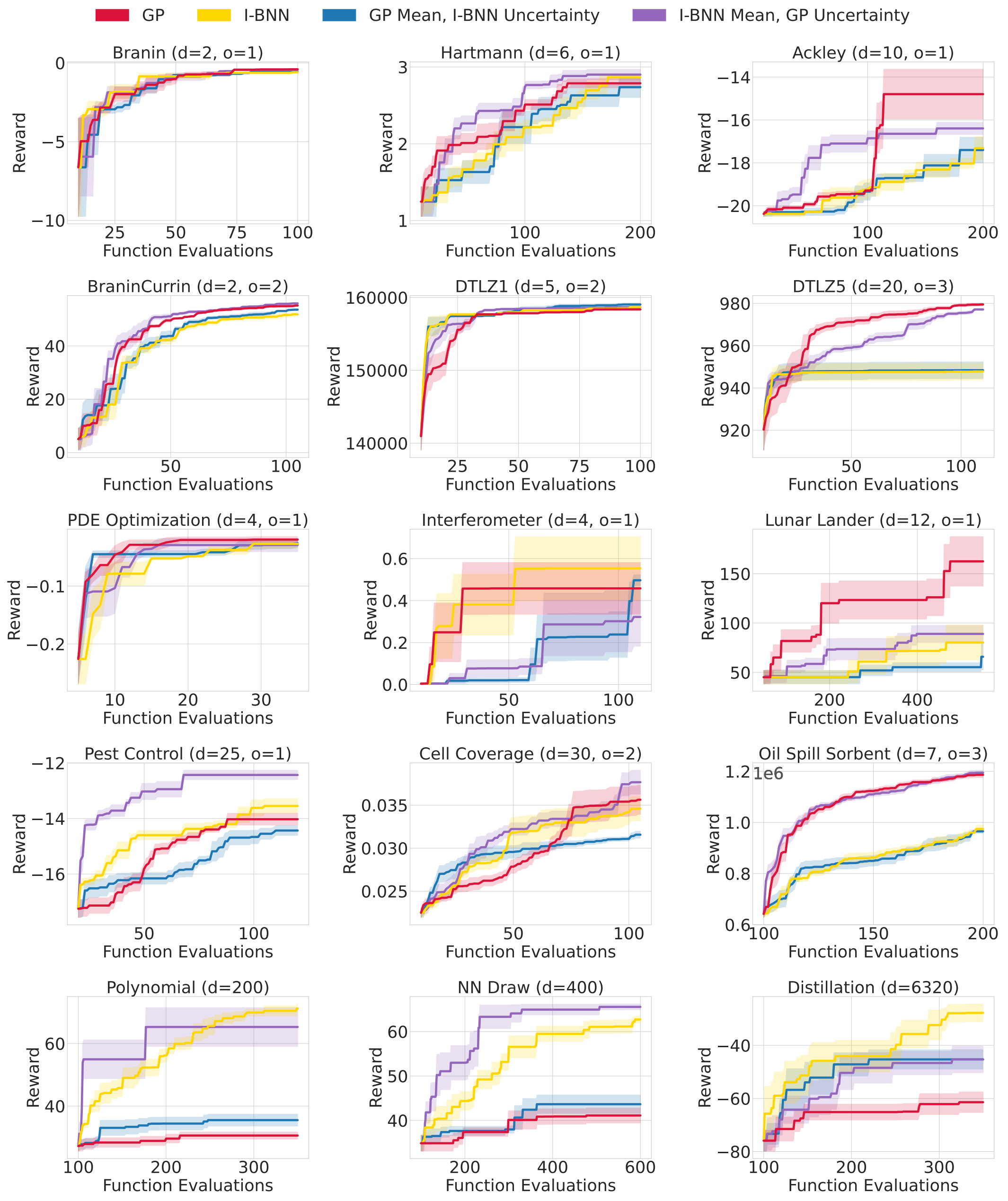


Figure A.4: The I-BNN-Mean GP-Uncertainty hybrid model outperforms GPs across a diverse set of problems, suggesting I-BNNs often provide better mean estimates. In the highest-dimensional problem of Knowledge Distillation (6,320 dimensions), I-BNNs outperform GPs in both the uncertainty and the mean estimates, suggesting that their non-Euclidean and non-stationary similarity metric is advantageous. We plot the mean and one standard error of the mean over 10 trials, d refers to the number of input dimensions, and o refers to the number of output dimensions.

D.3 Deep Ensembles

While deep ensembles often provide good accuracy and well-calibrated uncertainty estimates in other settings (Lakshminarayanan et al., 2017), we show they can perform relatively poorly for Bayesian optimization. For instance, when compared to other BNNs on benchmark problems such as BraninCurrin and DTLZ1, the maximum reward found by deep ensembles plateaus at a lower value than other surrogate models. These findings are also supported by results shown in Dai et al. (2022), where deep ensembles do not perform well in Bayesian optimization because they are unable to to explore the space effectively.

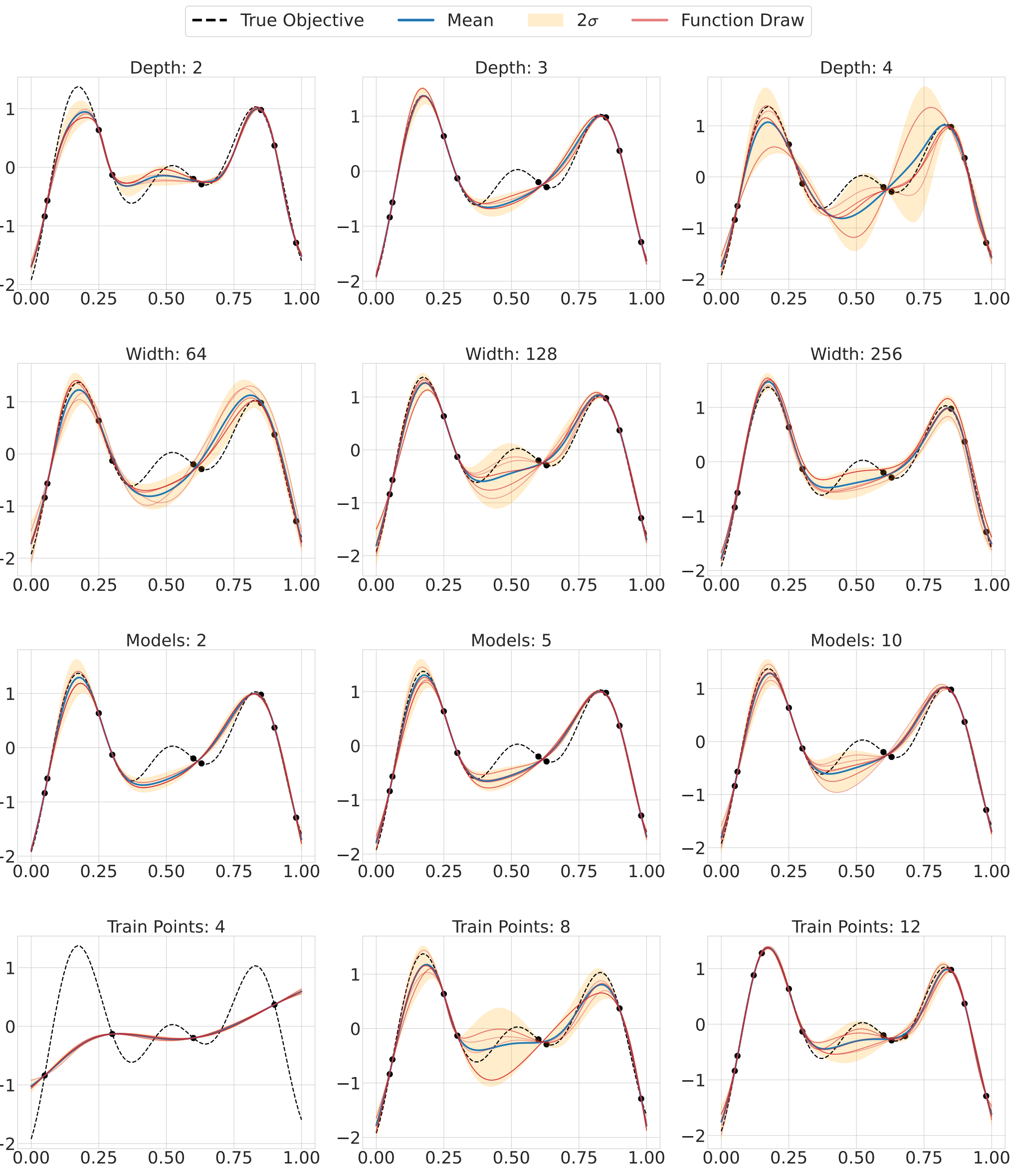
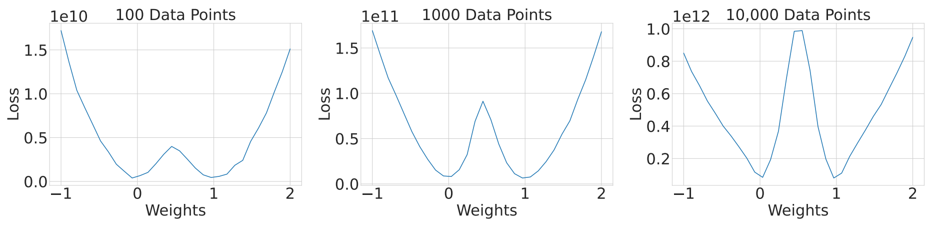
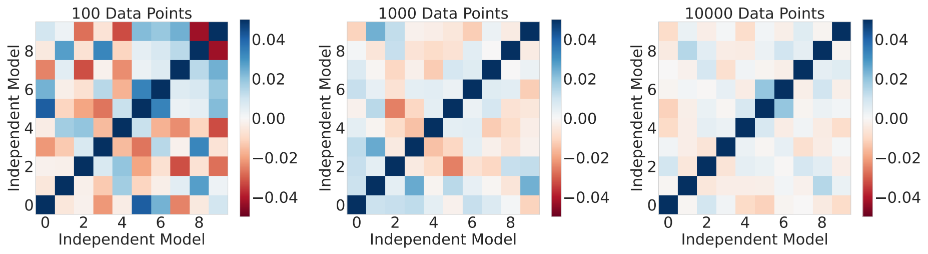


Figure A.5: We visualize the uncertainty of deep ensembles in various scenarios. For each set of experiments, we fix all other design choices with the following base parameters: depth = 3, width = 128, models = 5, train points = 9.



(a) Loss landscape



(b) Cosine similarity between models

Figure A.6: **With minimal training data, the loss landscape is relatively smooth, and separately-trained models are less diverse.** In the *top* figure, we train two models, corresponding to $x = 0$ and $x = 1$. We then linearly interpolate the weights between the two models to measure how the loss changes. As we increase the number of datapoints, the loss landscape becomes less smooth and models are able to find diverse basins of attraction. For the *bottom* figure, we train 10 models and plot the cosine similarity between weights. With less training data, the weights of the models are more related and the models are less diverse. We plot the results for DTLZ1 with 5 input dimensions and 2 output dimensions.

To further investigate the behavior of deep ensembles, we conduct a sensitivity study, varying the architecture, the amount of training data, and the number of models in the ensemble.

In Figure A.5, we see that smaller training sizes can paradoxically lead to less uncertainty with deep ensembles. A critical component in the success of deep ensembles is the diversity of its models. After training, each model falls within a distinct *basin of attraction*, where solutions across different basins correspond to diverse functions. Intuitively, however, in the low data regime there are fewer settings of parameters that give rise to easily discoverable basins of attraction, making it harder to find diverse solutions simply by re-initializing optimization runs. We consider, for example, the straight path between the weights of Model 1 and Model 2, and we follow how the loss changes as we linearly interpolate between the weights. Specifically, given neural network weights $\mathbf{w}_1$ from Model 1 and $\mathbf{w}_2$ from Model 2, and $\mathcal{L}(\mathbf{w})$ representing the loss of the neural network with weights $\mathbf{w}$ on the training data, we plot the loss $\mathcal{L}(\mathbf{w}_2 + (\mathbf{w}_1 - \mathbf{w}_2) * x)$ for varying values of x.

We share results in Figure A.6 for DTLZ1, a problem where deep ensembles performed poorly. We can see that in low-data regimes, although the loss between the two models does increase, the loss is significantly higher in other regions of the loss landscape. This behavior suggests that the basins are not particularly distinct, as the loss stays relatively flat between them, and thus less likely to provide diverse solutions. However, as we increase the amount of training data, the models are able to find more pronounced basins.

We also verify that the flatter regions correspond to a decrease in model diversity by measuring the cosine similarity between the model weights, and we see that in the low-data regime, models have more similar weights and therefore are less diverse.

In general, Bayesian optimization problems contain significantly fewer datapoints than where deep ensembles are normally applied. Standard Bayesian optimization benchmarks rarely exceed about 600 data points (objective queries), while in contrast deep ensembles are often trained on problems like CIFAR-10 which have 50,000 training points.

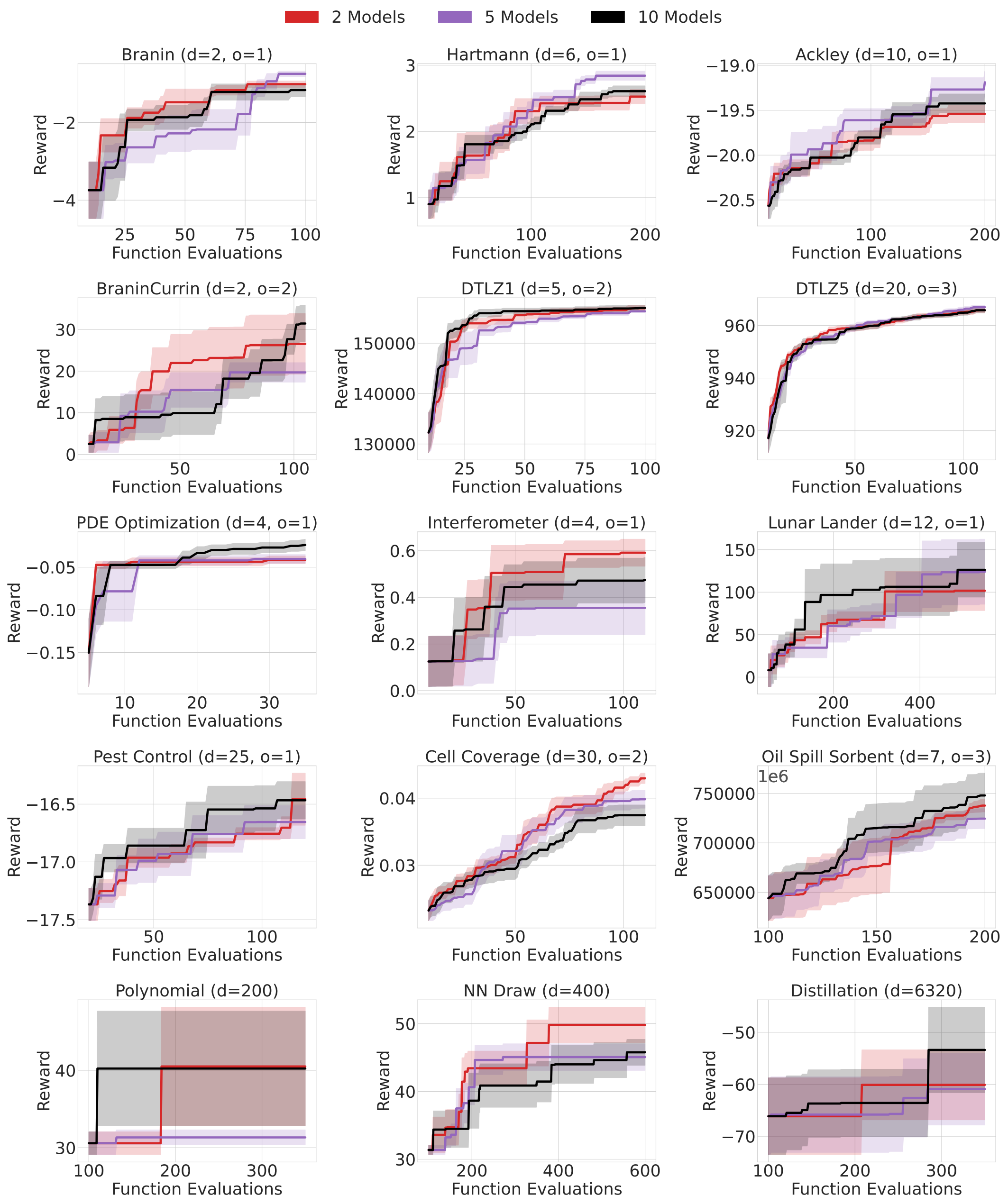


Figure A.7: We compare the behavior of ensembles with different numbers of models, and we find that the different ensembles perform similarly across many experiments, showing the robustness of our results to this hyperparameter. We plot the mean and one standard error of the mean over 10 trials, d refers to the number of input dimensions, and o refers to the number of output dimensions.

D.4 INFINITE-WIDTH BNNS

I-BNNS outperformed GPS in Bayesian optimization on a variety of high-dimensional problems, and we show results in Figure A.8. The first row of results corresponds to finding the maximum value of a random polynomial function, and the second and third rows show the results of maximizing a function draw from a neural network. For the neural network function draw benchmark, we experimented with many different architectures for the neural network to ensure that we had a diverse set of objective functions to maximize, as denoted in the title of each plot. In all cases, I-BNNS were able to find significantly larger values than GPS.

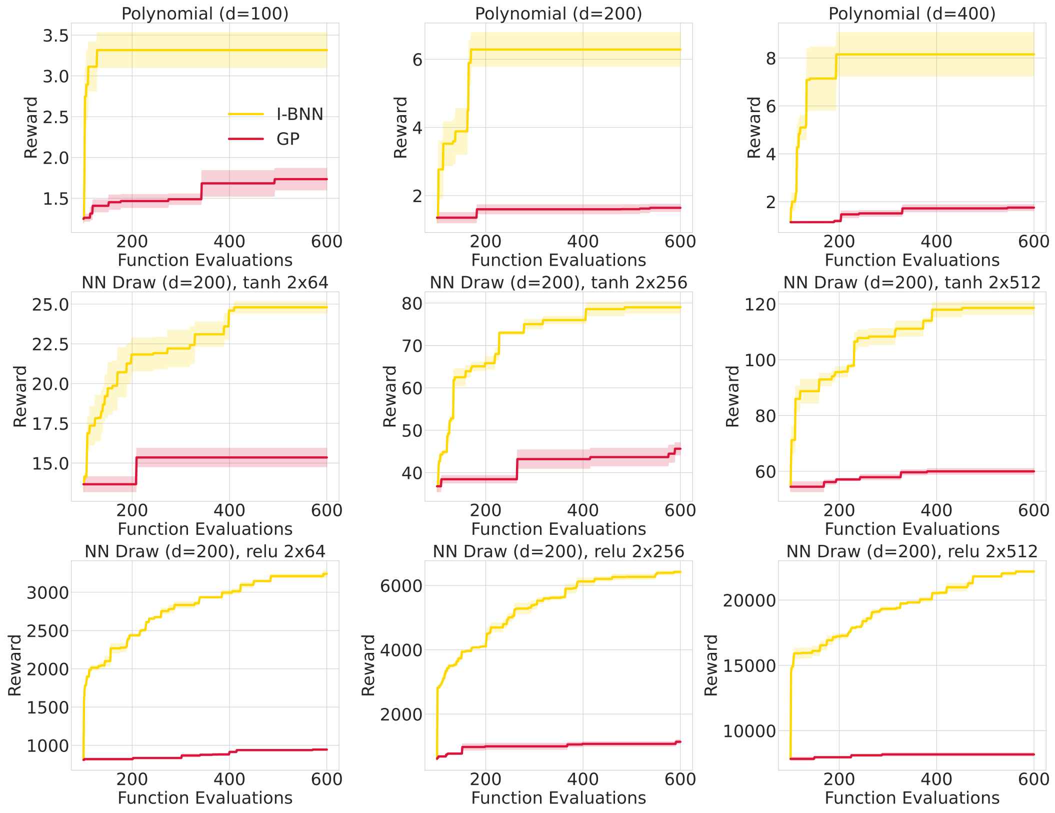


Figure A.8: **I-BNNS outperform GPS in high dimensions.** We plot the mean and one standard error of the mean over 3 trials, and d corresponds to the number of input dimensions. For the neural network draw benchmark, the architecture of the neural network which the function is drawn from is also included, where 2×64 means the network has 2 hidden layers of 64 nodes each.

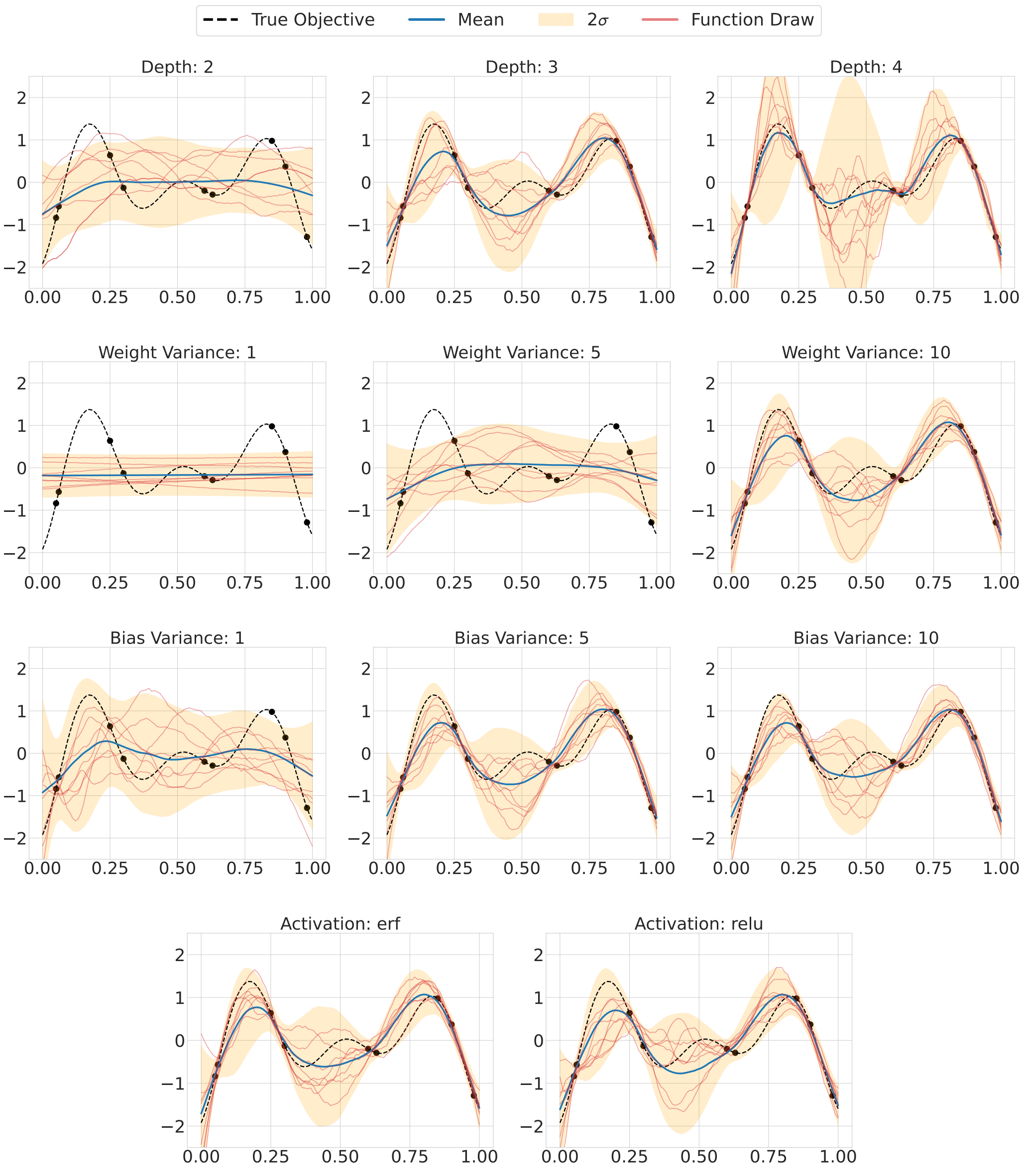


Figure A.9: We visualize how posterior predictive distribution of I-BNNs changes with different design choices. For each set of experiments, we fix all other design choices with the following base parameters: depth = 3, weight variance = 10, bias variance = 5, activation = relu.

In Figure A.9, we conduct a sensitivity analysis on the design of I-BNNs, showing how the posterior changes as we vary certain hyperparameters. Unlike standard GPs with RBF or Matérn kernels which are based in Euclidean similarity, I-BNNs instead have a non-stationary and non-Euclidean similarity metric which is more suitable for high-dimensional problems. Additionally, I-BNNs consist of a relatively strong prior, which is particularly useful in the data-scarce settings common to high-dimensional-problems.

Problem	Dim	GP MLL	I-BNN MLL
Branin	2	-3775.3	-1354.1
Hartmann	6	-1320.4	-1630.1
Ackley	10	-1463.6	-1824.3
Ackley	20	-1657.3	-2070.2
Ackley	50	-2788.9	-2316.4
Ackley	100	-4131.6	-2456.9
NN Draw	10	-3226.9	-1829.6
NN Draw	50	-15544.3	-2344.7
NN Draw	100	-39197.5	-2453.8
NN Draw	200	-35775.2	-2584.0
NN Draw	400	-48710.9	-2737.2
NN Draw	800	-63123.4	-2868.9
Distillation	6230	-2126658.7	-3059.8

Table A.1: **I-BNNs have larger marginal likelihoods than GPs on high-dimensional problems.** Here, we show the marginal log likelihood (MLL) of GPs and I-BNNs on various benchmark problems, and we estimate the MLL by sampling 1000 points randomly from the input domain.

To explore the suitability of the neural network kernel, we compare the marginal likelihood of GPs and I-BNNs on problems with various dimensions, and we report results in Table A.1. We see that I-BNNs and GPs have similar marginal likelihoods on problems with fewer dimensions, such as Branin and Hartmann, but as we increase the number of dimensions, the marginal likelihood of GPs becomes lower than that of I-BNNs. Interestingly, we see that the marginal likelihood of GPs does not decrease as quickly for Ackley as it does for the neural network draw test problem. This may be due to the neural network draw objective function having higher non-stationarity, so the standard GP kernel is less suitable for this problem compared to Ackley. For the knowledge distillation experiment, we see that the marginal likelihood of GPs plummets, and it was also not able to find high rewards for the problem as shown in Figure 5. Overall, I-BNNs have a higher marginal likelihood than GPs on high-dimensional problems, indicating that the I-BNN prior may be more reasonable in these settings.

D.5 HAMILTONIAN MONTE CARLO

We also explore the effect of the activation function on HMC. In Figure A.10, we see that the function draws from BNNs with tanh activations appear quite different from the function draws from BNNs with ReLU activations. The tanh draws are smoother and have more variation, while the ReLU draws seem to be more jagged. We also include results in Figure A.11 which compare the performance of ReLU and tanh activations in Bayesian optimization problems. We see that there does not appear to be an obvious trend, the optimal choice of activation function is problem-specific.

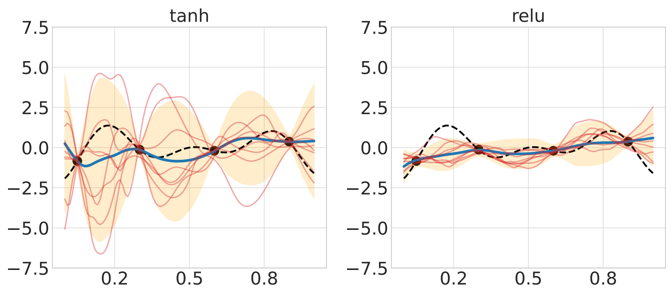


Figure A.10: The function draws of BNNs with tanh activations appear to be similar to the function draws from a GP. In contrast, the function draws from BNNs with ReLU are often jagged, and the network seems to deviate less from the mean.

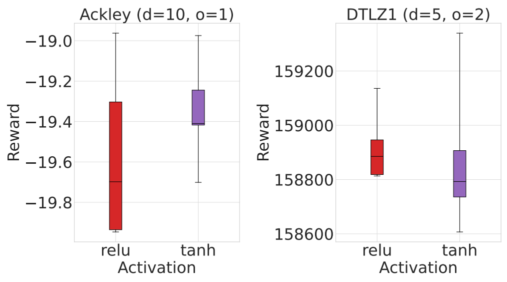


Figure A.11: In practice, ReLU and tanh activation functions can have comparable performance on synthetic functions.

D.6 Gaussian Processes

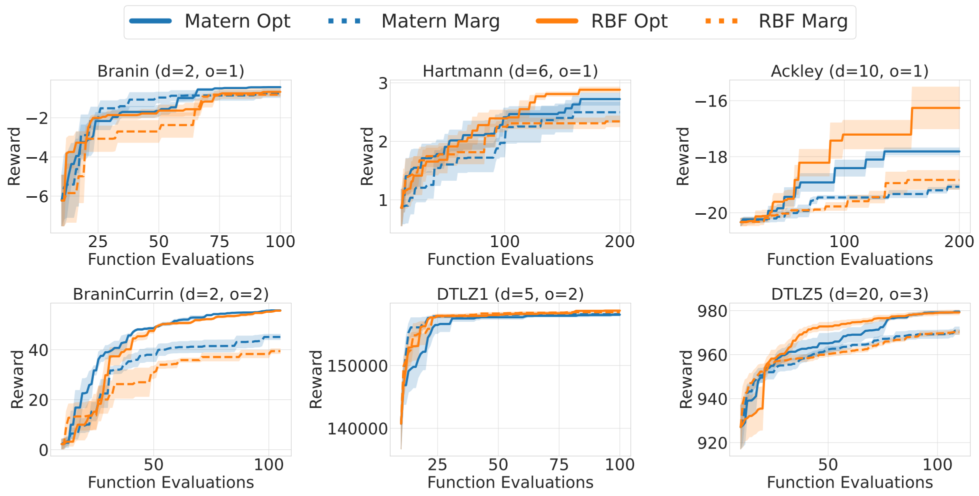


Figure A.12: **The point estimate for hyperparameter selection tends to improve performance on synthetic datasets.** The solid lines refer to hyperparameter optimization using a point estimate, while dashed lines correspond to fully Bayesian marginalization. We see that while the methods perform similarly on Branin, Hartmann, and DTLZ1, the point estimates perform slightly better for other datasets. The optimal choice of GP kernel is also problem-dependent. In many instances, Matérn and RBF perform similarly, although the point estimates for Matern outperform RBF on Ackley. We plot the mean and one standard error of the mean over 10 trials, d refers to the number of input dimensions, and o refers to the number of output dimensions.

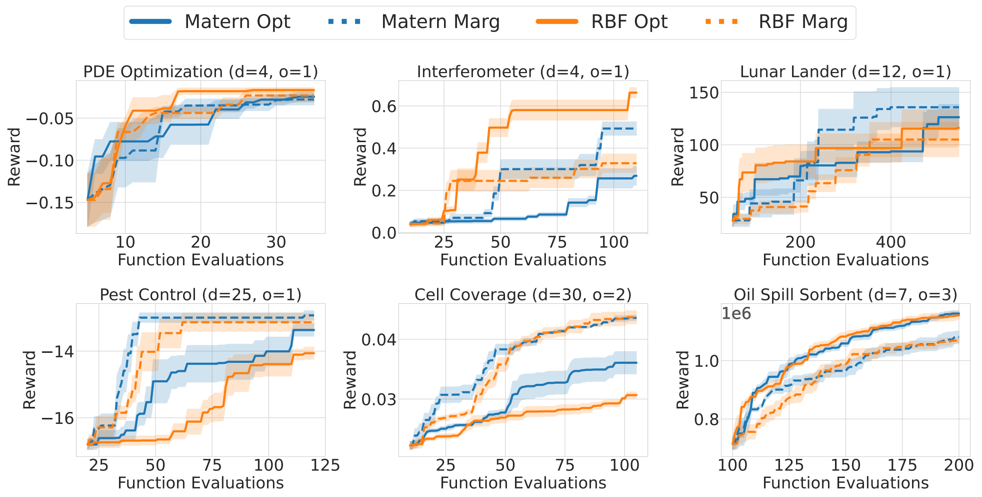


Figure A.13: **The method of hyperparameter selection greatly impacts the performance of GPs on real-world datasets.** The solid lines refer to hyperparameter optimization using a point estimate, while dashed lines correspond to fully Bayesian marginalization. We see real-world datasets often find marginalization of the hyperparameters to be more effective. The optimal choice of GP kernel is also problem-dependent. In many instances, Matérn and RBF perform similarly, although the point estimates for Matérn outperform RBF on discrete problems such as Pest Control and Cell Coverage. We plot the mean and one standard error of the mean over 10 trials, d refers to the number of input dimensions, and o refers to the number of output dimensions.

D.7 Deep Kernel Learning

Rather than using the marginal likelihood to optimize parameters for DKL, we can also use the conditional marginal likelihood (Lotfi et al., 2023). We see in Figure A.14 that there is no clear preference for using the marginal likelihood (ML), which we use through all other experiments in the paper, or conditional marginal likelihood (CML) for our problems.

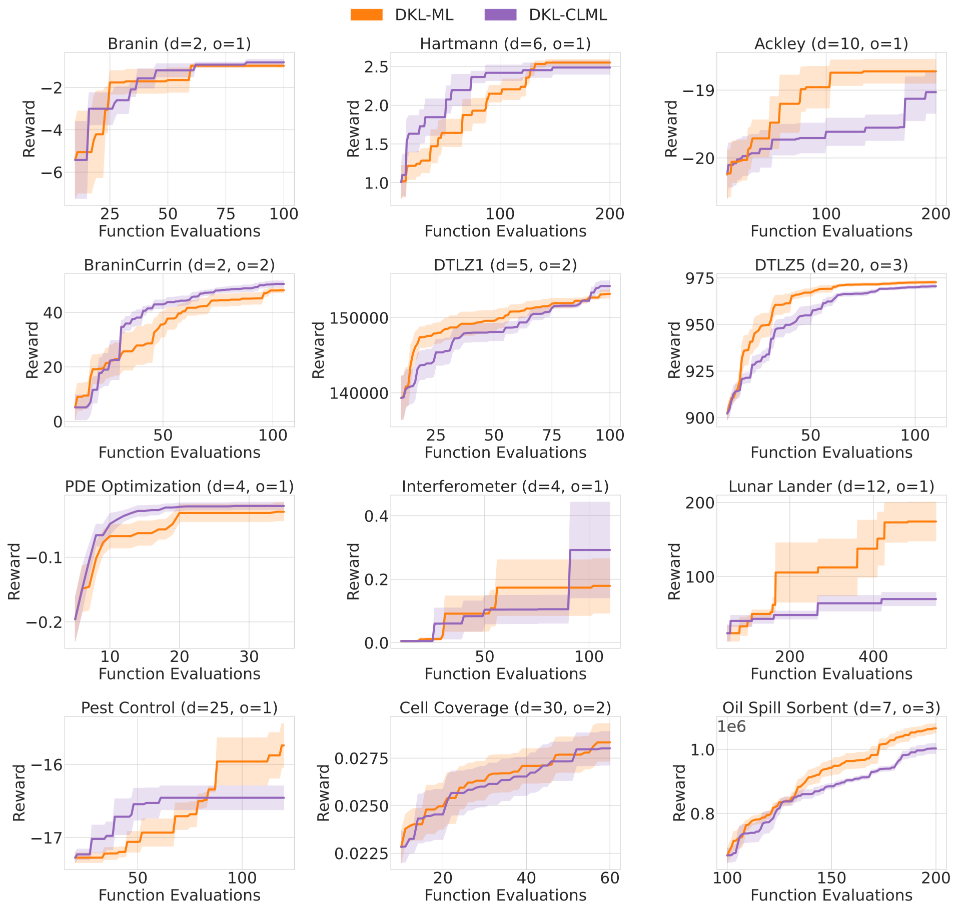
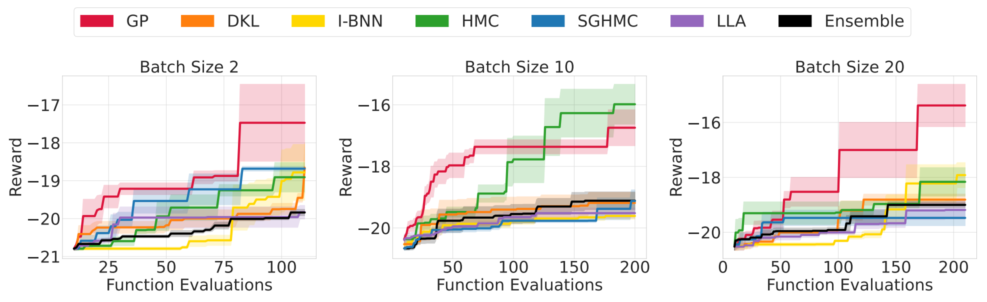
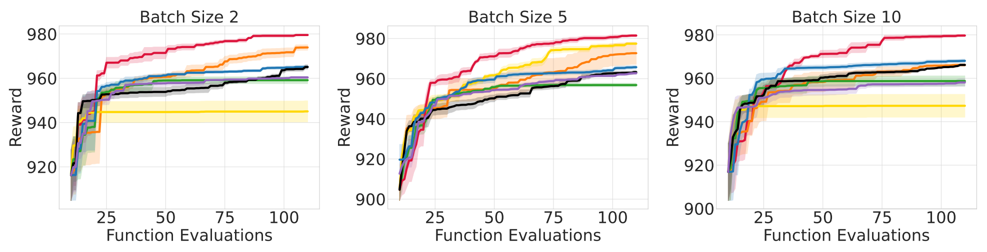


Figure A.14: There is no clear preference for using the marginal likelihood (ML) or conditional marginal likelihood (CML) to optimize the parameters of DKL

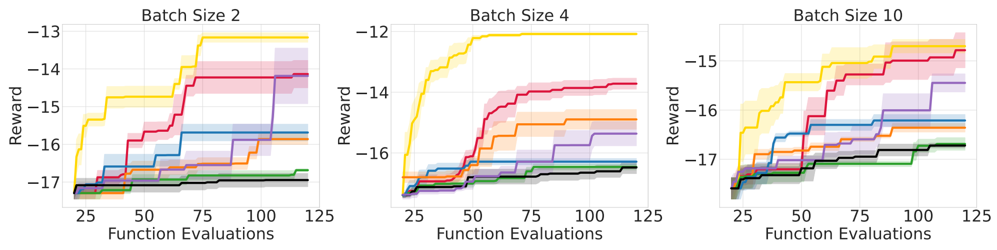
D.8 Acquisition Batch Size



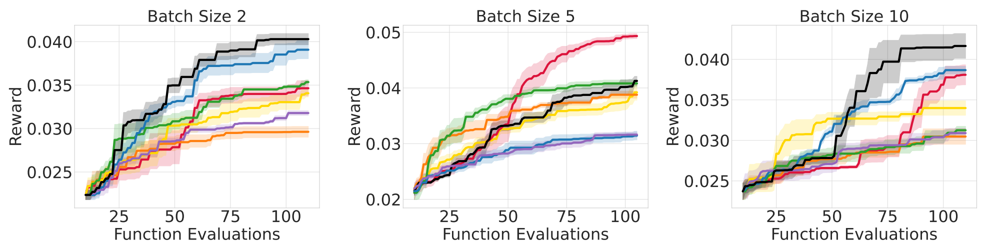
(a) Ackley (d=10, o=1)



(b) DTLZ5 (d=20, o=3)



(c) Pest Control (d=25, o=1)

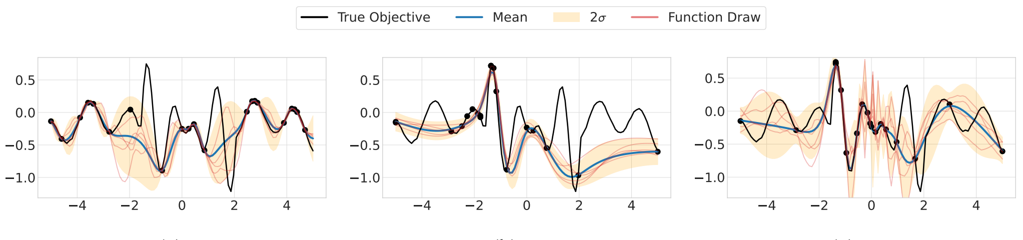


(d) Cell Coverage (d=30, o=2)

Figure A.15: The batch size of the acquisition function impacts the performance of the models, but the general trends remain similar. The relative performance of the models are similar across varying batch sizes, with a few notable exceptions. The I-BNN seems to plateau on DTLZ5, and is unable to find high rewards. GPs also seem to perform significantly better on Cell Coverage with a batch size of 5 compared to other sizes. For each benchmark, we include d for the number of input dimensions, and o for the number of objectives. We plot the mean and one standard error of the mean over 10 trials.

D.9 Limitations of Gaussian Process Surrogate Models

Due to their assumptions of stationarity as noted in Section 4.3, GPs struggle in non-stationary settings. In Figure A.16, we compare the posterior distribution from a GP with the posterior from different BNN surrogate models. We show results after 20 iterations of Bayesian optimization, starting with an initial point of $x = 0$. The function we wish to maximize is non-stationary: the function has greater variance between -2 and 2, and there is also a slight downward trend. We see that due to their stronger assumptions, GPs are not able to find the true global maximum of 0.8, instead getting suck in local optima. In contrast, HMC and I-BNNs are able to find the global maximum within 20 iterations.

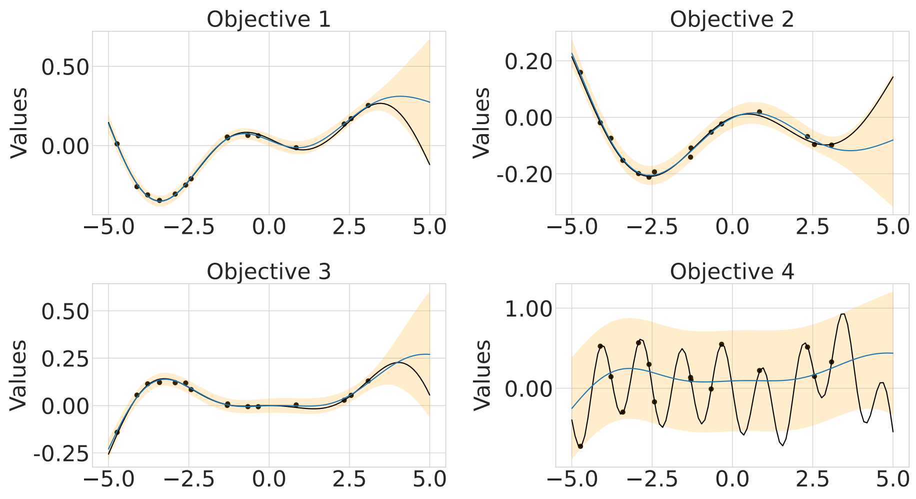


(a) GP (b) HMC (c) I-BNN

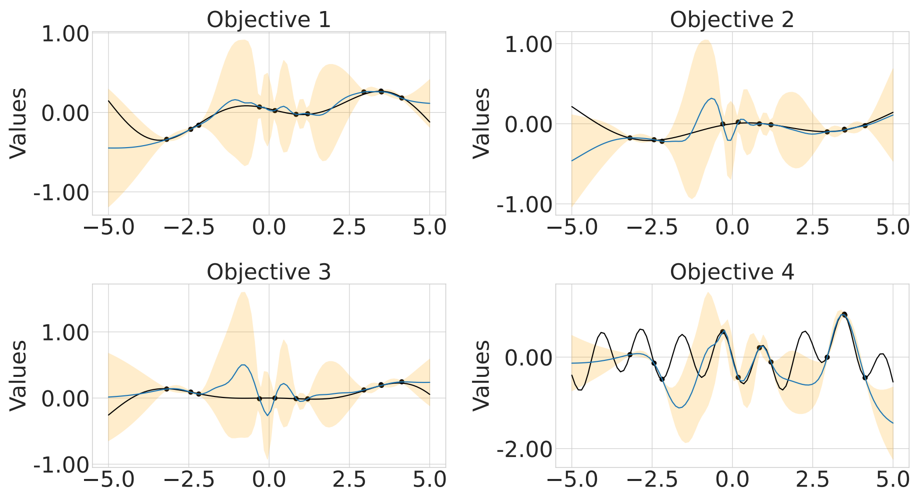
Figure A.16: **GPs struggle to find the global maximum for non-stationary functions**. After 20 function evaluations, the GP does not accurately model the true function around $x = -1$ because it is unable to account for the sudden increase in scale due to its assumption of stationarity. BNNs do not suffer from the same pathologies and are able to find the true maximum.

An additional limitation of GP surrogate models, which we demonstrate in Figure A.17, is its performance on multi-objective problems in Bayesian optimization. Although GPs have been successfully extended to a wide range of multi-objective problems, in the interest of making the approaches scalable, there are many assumptions placed onto the kernel. In the most naive setting, we can model each objective independently. While this approach is convenient, it completely ignores the relationship between objective values and has no notion of shared structure, so it is unable to take advantage of all of the information in the problem. Multi-objective covariance functions can also be decomposed as Kronecker product kernels. While this approach can have significant computational advantages compared to modeling the full covariance function, it requires each objective to itself be modeled with the same underlying kernel. Thus, this method of modeling multiple objectives will fail to capture the nuances of each particular objective when the functions have differing properties.

In Figure A.17, we show the result of twenty iterations of Bayesian optimization over a synthetic multitask example, where we care about optimizing over the fourth function but provide additional information through the other three datasets. We use the GP with Kronecker product kernels to model the multiple objectives. In our experiment, although the GP is able to learn the proper length scale and variance over the three additional functions with similar length scales, it struggles to accurately fit the fourth objective. Because the GP is unable to account for the differences between the four functions, it does not find the global optimum. Unlike GPs, BNNs are not restricted to strict covariance structures and are able to produce well-calibrated uncertainty estimates in multi-objective settings. The BNNs are able to accurately fit all four functions, including the fourth objective function which has a much smaller length scale compared to the others.



(a) GP



(b) HMC

Figure A.17: **GPs have a hard time finding the global maximum in multi-objective settings**. Multi-task GPs learn one length scale across all objectives, which may not be suitable for many datasets. In this example, it does not find the global minimum in the 4th objective because it treats the shorter length scale as noise. In contrast, BNNs are able to appropriately model the uncertainty for the 4th objective and find its true minimum.

D.10 Large Number of Function Queries

To further accentuate the distinctions between BNNs and GPs, we experiment with a larger number of function queries. Specifically, we look to maximize a function with 200 input dimensions drawn from a fully connected neural network with 5 layers and 256 nodes per layer. We expect this function to have a high degree of non-stationarity, making it difficult for standard GPs. We start with 2000 initial function queries and end the experiment at 3000 function queries. We share results in Figure A.18.

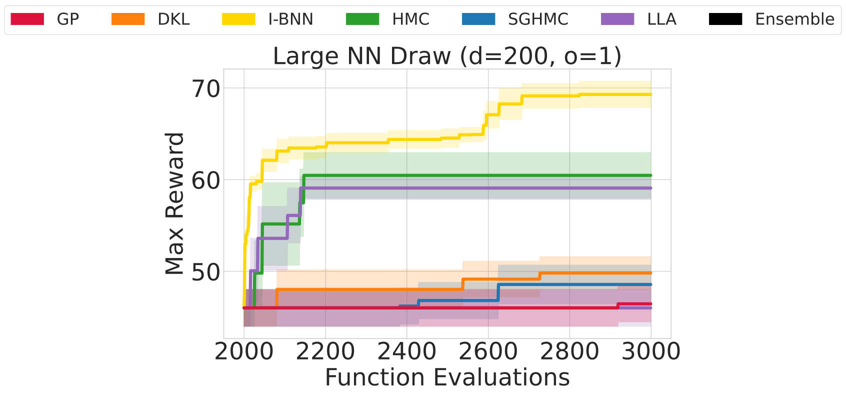


Figure A.18: BNNs outperform GPs when there are a large number of function queries.

This experiment reveals several valuable findings: (1) I-BNNs remain competitive relative to the alternatives; (2) BNN surrogates start to perform more effectively, able to leverage the additional data for representation learning; (3) deep ensembles, in particular, are greatly improved with more data, in line with our explanation that the relative poor performance was due to an inability to find diverse models corresponding to posterior modes when there is limited data. At the same time, we note most Bayesian optimization problems do not have many objective queries, since Bayesian optimization is often found to be most valuable when the objective is expensive to query. In this light, perhaps (1) is the most valuable of the findings, since it shows consistency of the relatively strong I-BNN surrogate. In the future, we might expect I-BNNs to become a mainstream surrogate model for Bayesian optimization.

D.11 Runtime

While inference time is relevant for the comparison of surrogate models, in many real-world Bayesian optimization scenarios, the most expensive computation often lies in the querying of the objective function, which may include actions such as synthesizing a new material, training a large neural network to convergence, etc. For these scenarios, the quality of the surrogate model uncertainties may be much more important than the cost of inference. For completeness, in rare instances where inference-time is a relevant consideration, we provide wall-clock times of all surrogate models across our experiments in Table A.2, and we compare the performance of the surrogate models within a fixed time budget in Figure A.19

Problem	GP	DKL	I-BNN	HMC	SGHMC	LLA	Ensemble
Branin	2.62	124.55	3.97	214.40	142.21	19.29	190.24
Hartmann	8.20	123.00	7.79	663.22	381.57	49.25	413.50
Ackley	4.28	263.78	4.69	141.66	201.48	28.29	225.41
BraninCurrin	14.17	198.80	19.76	237.71	365.86	61.09	432.94
DTLZ1	8.25	117.02	10.82	183.74	127.54	21.97	194.68
DTLZ5	21.50	308.12	22.18	194.81	147.07	38.99	231.96
PDE	240.48	512.98	239.23	456.48	628.14	312.40	981.50
Interferometer	16.09	939.49	17.97	690.07	540.59	83.61	944.87
Lunar Lander	458.96	3140.30	621.20	3118.65	684.61	802.13	829.44
Pest Control	3.84	268.06	5.43	66.18	219.08	27.87	253.37
Cell Coverage	9.50	593.09	13.26	101.23	181.11	35.00	243.17
Oil Spill	37.51	300.20	44.81	291.53	350.30	124.71	438.89
Polynomial	16.28	673.40	20.59	211.94	486.75	106.77	540.41
NN Draw	232.48	1458.11	391.95	896.34	451.34	799.88	1124.63
Distillation	4719.40	3982.80	4937.27	4282.89	299.40	3223.16	3007.93

Table A.2: Wall-clock time in seconds of one trial of each experiment, with experiment details specified in Appendix C. We record the mean time over 10 trials.

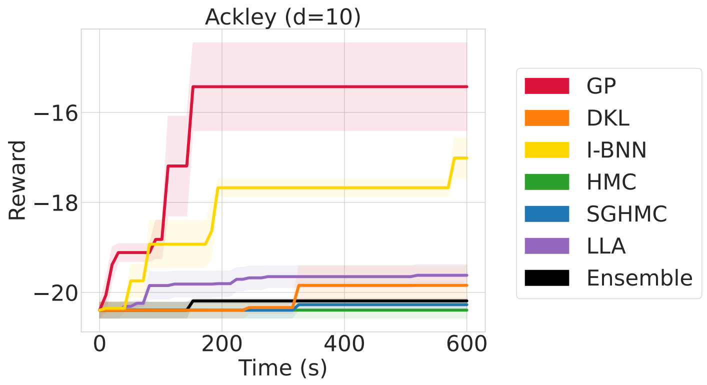


Figure A.19: For objective functions which are very inexpensive to query, like Ackley, we find GPs and I-BNNs to outperform other surrogate models given a fixed amount of time. However, for the many Bayesian optimization problems instead which expensive to query, and the total time would be dominated by the function query instead of inference time. We record the mean and standard error over 10 trials.